Universidad
Carlos III de Madrid
www.uc3m.es

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Grado en Ingeniería Telemática
(2009-2016)

“Diseño y despliegue de una red definida por software sobre máquinas virtuales”

Trabajo Fin de Grado

Autor:

José Manuel Frías Pérez

Tutor:

Jaime José García Reinoso

Agradecimientos

Quiero dar las gracias a todas aquellas personas que, de una forma u otra, me han acompañado durante todo este camino.

En primer lugar a mi familia, porque sin ellos nada de esto hubiera sido posible, por darme la oportunidad de estudiar una carrera y por su apoyo incondicional.

A mi tutor, Jaime, por haberme guiado durante estos meses, haber respondido todas mis dudas y facilitado el desarrollo de este Trabajo Fin de Grado.

Agradecer también a todos mis compañeros de universidad, a Nacho por esas tardes de café y ofi hasta las tantas, a Nuria por acompañarme durante gran parte de esta carrera, pero en especial al equipo SGTK, Sergi, Mario, Guti y Giby, por haber formado parte de todo esto desde el principio, por esos buenos y malos momentos y por haber hecho más ameno el camino.

A mi amiga Teresa por su inestimable ayuda.

Y por último y no por ello menos importante, a Ana, por haber sido capaz de sacarme una sonrisa cada vez que lo necesitaba y por apoyarme día a día.

RESUMEN

Durante los últimos años los temas más candentes relativos a las redes de comunicaciones han sido *Software Defined Networking* (SDN) y *Network Function Virtualization* (NFV).

Tradicionalmente las funcionalidades de red se implementan en el hardware de un dispositivo dedicado: *router*, *switch, etc,* haciendo que las redes evolucionen lentamente y se encuentren limitadas en funcionalidades; para mejorar esto, se introdujo SDN.

Debido a la actual necesidad de las empresas de virtualizar servidores y trabajar con MVs (Máquinas Virtuales) de forma ágil, se han empezado a desarrollar parte de las funciones hardware, de los dispositivos mencionados anteriormente, en software alojado en un servidor o máquina virtual, facilitando así la automatización de servicios y aplicaciones.

Todo esto nos lleva al desarrollo de SDN y los problemas que puede abordar:

- Provisión de QoS y seguridad
- Implementación y funcionalidad de ampliación de red más sencilla.
- Mayor aprovechamiento de los recursos de red.
- Reducir el OpEX y el CapEx
- Funcionalidades de red más dinámicas en función de la evolución del software.
- Reducir la complejidad.

SDN se encarga de desacoplar las funciones del plano de control y el reenvío de la red, permitiendo al control de la red hacerse programable y ajustar dinámicamente el tráfico para satisfacer las necesidades cambiantes.

En las redes, el plano de datos se encarga de usar las tablas de enrutamiento configuradas por el plano de control.

En la arquitectura de las redes definidas por software, el control está centralizado. Los controladores SDN basados en software mantienen una visión global de la red permitiendo administrar y optimizar los recursos de forma dinámica mediante software basado en estándares abiertos que no limitan las operaciones a realizar.

Si hablamos de SDN es obligatorio hablar de OpenFlow [9], un estándar abierto que permite probar protocolos experimentales en redes que utilizamos a diario. OpenFlow se añade como una característica en los *switches* Ethernet comerciales, *routers* y puntos de acceso inalámbricos. Actualmente los principales proveedores lo están implantando es sus *switches*.

El funcionamiento básico de SDN consiste en separar el plano de datos del plano de control. El plano de datos continúa en el hardware del *switch*, mientras que las decisiones de enrutamiento de alto nivel son trasladadas a un controlador. El *switch* y el controlador OpenFlow se comunican con dicho protocolo, el cual se encarga de definir ciertos tipos de mensajes específicos para este tipo de conexiones.

Todas estas ventajas, innovaciones y el auge actual de SDN han motivado el desarrollo de este Trabajo fin de Grado.

Este proyecto consiste en el desarrollo de un guion de prácticas que será utilizado para enseñar SDN a estudiantes de universidad, debido a la importancia que tiene en el marco de las telecomunicaciones actualmente y su futuro prometedor.

Para la consecución de dicho objetivo se ha propuesto el diseño y despliegue de una red definida por software sobre máquinas virtuales. Las tareas llevadas a cabo durante este TFG son las siguientes: estudio de SDN y sus principales aplicaciones, estudio del estándar OpenFlow, estudio de la herramienta Mininet [3] para virtualización de redes, investigación sobre la interconexión de la red creada en Mininet y un controlador alojado en una máquina virtual externa, aprendizaje de los tipos de aplicaciones y funcionalidades que pueden ser utilizadas en el controlador SDN Opendaylight [10].

La primera fase consiste en el estudio de SDN y posibles controladores a utilizar.

Entre los distintos controladores existentes: Floodlight, Ryu, NOX y Opendaylight, decidimos escoger Opendaylight por la mayor cantidad de documentación y ejemplos disponibles, y porque es un controlador implementado en Java, lo cual facilita la ejecución de programas y el despliegue del entorno de trabajo.

Una vez elegido el controlador, desplegamos el entorno de trabajo utilizando Lubuntu 14.04.3 y haciendo uso de la herramienta de virtualización Virtual Box. Lubuntu es un sistema operativo que no requiere de muchos recursos y es ligero, permitiendo desarrollar el TFG en un único portátil.

Esto último es un requisito indispensable para poder emplear este Trabajo Fin de Grado como recurso docente en un futuro, facilitando a la universidad las máquinas virtuales ya configuradas y funcionales.

Una vez instalado el sistema operativo descargamos el controlador Opendaylight y lo arrancamos. Al arrancar el controlador probamos algunos comandos del CLI e instalamos los *features* necesarios para nuestro TFG.

Por otro lado, descargamos la máquina virtual que proporciona la página web de Mininet y comenzamos el estudio del funcionamiento de los comandos básicos y las diferentes topologías predeterminadas que se pueden desplegar, así como otras funciones que nos serán útiles en el futuro, como puede ser la creación de un servidor web en uno de los host de la topología y realizar una petición GET desde otro de los host, para ver los paquetes intercambiados.

Una vez desplegadas las dos máquinas debemos conectar ambas de tal forma que Mininet utilice el controlador remoto instalado y así poder, desde el controlador, añadir los flujos que necesitemos y visualizar la topología creada desde la interfaz web proporcionada por Opendaylight.

Tras configurar la interconexión de ambas máquinas y realizar pruebas con topologías predeterminadas de Mininet, pasamos a probar topologías nuevas y personalizadas, utilizando el editor de topologías que nos proporciona la máquina virtual.

La topología utilizada en esta parte de la práctica fue desarrollada en un fichero Python, y consiste en tres *switches* interconectados, con un host en cada uno de ellos. La complejidad de la topología no debe ser necesariamente elevada ya que las pruebas realizadas en una topología sencilla son fácilmente extrapolables a una más compleja, añadiendo elementos a la red desplegada en Mininet.

Tras la correcta conexión y la prueba de existencia de conectividad, mediante el comando ping entre los host desplegados por Mininet, instalamos un flujo sencillo en uno de los *switches*.

A continuación debemos desarrollar los flujos que queremos utilizar, para ello hay que conocer la forma de trabajo de Opendaylight.

Opendaylight permite tres tipos diferentes de programación: Java API, DOM API y REST API, todas ellas “traducibles” por la Service Abstraction Layer que explicaremos en futuros puntos de este TFG. Además existen dos formas de trabajar con este

controlador, instalando Bundles en el framework OSGi del propio controlador o mediante instrucciones REST que se ejecutan de forma remota.

El primer flujo está configurado en un fichero XML y utiliza la REST API del controlador Opendaylight, se ejecutará de forma remota, y su funcionalidad es descartar los paquetes del host 1 (h1) al host 2 (h2), para ello se utiliza un filtrado MAC de origen y destino. El flujo lo instalamos en el *switch* (s1) de la topología mediante una petición PUT a la dirección IP de s1 desde el controlador. Para comprobar el correcto funcionamiento del flujo instalado de forma sencilla, miramos las tablas de flujos instalados en el s1 y vemos cómo al hacer un *ping*, el número de paquetes recibidos del flujo instalado comienza a aumentar, y en el terminal del *ping* vemos que h2 no es alcanzable.

Una vez desplegado el escenario del TFG y comprobado el correcto funcionamiento de flujo configurado en el fichero XML, correspondiente a la primera parte de la práctica, decidimos crear un proyecto Maven en el controlador y proceder al desarrollo de la segunda parte de la práctica.

Maven es un gestor de proyectos basado en el concepto POM (Project Object Model), y nos permite crear proyectos, reportes y documentación a raíz de cierta información.

Como hemos mencionado anteriormente, Opendaylight es un conjunto de complementos OSGi, clases Java y otra serie de recursos y manifiestos que funcionan sobre el *framework* OSGi. Además, está basado en Maven, por ello lo mejor para crear nuestro proyecto será seguir la estructura que esta herramienta nos facilita.

La creación del proyecto consiste en generar con la herramienta Maven una serie de directorios y ficheros compatibles con el controlador, de tal forma que al iniciar el controlador, la aplicación Java que nosotros queramos instalar sea utilizada por él.

El proyecto consiste en una aplicación Java northbound instalada en el controlador, que arranque al inicio del mismo y a través de la interfaz web de Opendaylight poder añadir flujos que filtren y reenvíen los paquetes en función del protocolo indicado, del mismo modo el flujo podría ser instalado en el *switch* como en la primera parte de la práctica.

La aplicación Java que vamos a utilizar ha sido desarrollada por el grupo SDNHub, de código abierto.

Decidimos utilizar una aplicación ya desarrollada dado que el objetivo de este TFG es enseñar SDN y no a programar en Java. De este modo adecuamos la aplicación *open-source* a nuestras necesidades modificando las líneas y los ficheros que fueron necesarios para que la aplicación funcionase con nuestros requerimientos y necesidades.

Una vez instalada la aplicación comprobamos su correcto funcionamiento realizando pruebas con diferentes protocolos e instalándola en diferentes *switches* y de diferentes formas, utilizamos comandos *curl* como en la primera parte de la práctica y empleamos la interfaz web que nos proporciona uno de los complementos de Opendaylight (DLux) para insertar los flujos de una forma más visual; además estudiamos los mensajes que aparecen en el *log* del controlador al realizarse los inicios de conexión, su finalización, el envío de un flujo, etc.

Una de las pruebas realizadas es desplegar un servidor web en uno de los host y realizar un petición desde otro. Si configuramos el flujo para que filtre el protocolo HTTP, veremos cómo se incrementan los paquetes recibidos en el flujo instalado.

Por tanto en la práctica a desarrollar tenemos dos partes, una más sencilla en la que explicamos el funcionamiento de la REST API y generamos un fichero de configuración para descartar los paquetes que van de un host a otro, y por otro lado tenemos una aplicación Java, instalada en el *framework* OSGi del controlador, y una serie de ficheros *JSON*, *pom* y *manifest* de configuración, algo más complejos y extensos que la primera parte.

En conclusión, la primera parte es un flujo sencillo pero muy visual para comenzar a conocer el funcionamiento de SDN y cómo interactúan el controlador y las topologías desplegadas en Mininet. La segunda parte, por el contrario, es algo más compleja y sirve para estudiar y entender SDN y su potencial más a fondo, utilizando el gestor de proyectos con el que trabaja Opendayligh, cómo se instala una aplicación en el controlador, cómo se ejecuta y cómo se pueden añadir, modificar o eliminar flujos desde la interfaz web proporcionada por el *feature* DLux.

ABSTRACT

Software Defined Networking (SDN) and *Network Function Virtualization* (NV) have been two of the most relevant topics when talking about communication networks in the past years.

Traditionally, network functionalities are implemented in the hardware of a dedicated devices: router, switch.etc, making networks evolve slowly and limiting them in terms of functionalities. SDN was introduced as a way to improve these limitations.

Currently, companies need to virtualize servers and work with Virtual Machines (VMs) at an increasing speed. Therefore, part of the hardware functionalities have started to be moved from dedicated devices to software hosted in servers or virtual machines. This has facilitated the automation of servers and applications.

But before advancing further, it is convenient to understand the development of SDN and the problems this improvement is able to solve:

- Provision of QoS and security
- Easier implementation and functionality of network's growth
- Increased usage of network resources
- Reduce OpEx and CapEx
- More dynamic network functionalities in terms of software evolution
- Reduction of complexity

SDN is in charge of unlinking the functionalities of the network control and forwarding planes, allowing the network control to be programmed and to dynamically adjust the traffic to satisfy the changing needs.

Inside the network, the data plane is responsible for transferring a datagram from a specific entry to its respective exit. The control plane decides the route to be followed by the datagram using the routing tables.

The architecture of networks defined by software is centralized. The software-based SDN controllers keep a global view of the network facilitating the management and optimization of resources in a dynamic manner, using software based on open standards that do not limit the operations to be made.

Talking about SDN makes it necessary to talk about OpenFlow, an open standard that allows trying experimental protocols in daily used networks. OpenFlow is added as a characteristic in commercial Ethernet switches, routers and wireless access points. Currently, the main providers are implanting it on their switches.

The basic functioning of SDN consists on separating the data from the control plane. The data plane remains in the switch's hardware while the high-level routing decisions are transferred to a controller. The switch and the Openflow Controller communicate with the aforementioned protocol, in charge of defining certain specific of messages for this type of connections.

All these SDN-related facts and improvements have sparked our interest on the subject, and therefore motivated its study and the subsequent elaboration of this Bachelor Thesis.

The main objective of this Final Bachelor Thesis (TFG) is to design and display a software-defined network that can be used as a lab test in University. The lab test guide we will elaborated to teach SDN to university students due to its relevance in the telecommunications area and its promising future.

In order to achieve our purpose, we will take the following steps: study SDN and its main applications, study the OpenFlow standard, study Mininet (network virtualization tool), research about the network interconnection created with Mininet and a controller hosted in an external virtual machine; learning about other types of applications and functionalities that can be used in the SDN Controller Opendaylight.

In the first part, we will study SDN and possible controllers to use. Among the existing controllers (Floodlight, Ryu, NOX and Opendaylight), we decided to choose Opendaylight because there was more information available. Additionally, it is a Controller implemented in Java, that therefore facilitates the execution of programs and the deployment of the environment.

Once the controller has been chosen, we deploy the environment using Lubuntu 14 and the Virtual Box tool. Lubuntu is a light operating system that does not require many resources, allowing us to carry the project in a single laptop. The latter is a requisite for the use of this Final Thesis as teaching material in the future as it provides the university with functioning virtual machines.

Once the operating system has been installed, we download the Opendaylight controller and we turn it on. When it is functioning we try several CLI commands and we install the necessary features for our work.

Separately, we download the virtual machine provided by Mininet's website and we begin studying the functioning of basic commands and predetermined topologies. We also try other functionalities that could be useful in the future: creation of a web server in one of the topology hosts and make a GET petition from a different host in order to see the exchanged packages.

After both machines have been deployed we should connect them in a way that Mininet uses the previously installed remote controller. This will let us add the necessary flows by using the topologies editor that comes with the virtual machine.

The topology used in this part was developed in a Phyton file and consists of three interconnected switches with a host on each. The topology does not need to be complex as our trials, done in a simple topology, are easily extrapolated to a more complex one just by adding elements to the network deployed in Mininet.

Once the connection is right and we have tested it with the ping command between the hosts deployed by Mininet, we install a simple flow in one of the switches. Aferwards, we should develop the flows we want to use. To do so we need to how Opendaylight works.

Opendaylight allows three different programming types: Java API, DOM API and REST API, all of them "translatable" by the Service Abstraction Layer that we will later explain. Additionaly, we can work with the Controller in two different ways: installing bundles in the OSGi framework of the controller itself or using REST instructions which are remotely executed.

The first flow is configured in a XML file and uses the REST API from the Opendaylight controller. It will be remotely executed with the aim of dropping the packages from Host 1 (h1) to Host 2 (h2). The drop action can be done thanks to a MAC origin and end filter. The flow is installed in the switch (s1) from the topology through a PUT petition to the s1 IP address from the controller. To ensure the right functioning of the installed flow, we look at the flow tables installed in s1 jut to realize how after the pinging, the amount of received packages starts to rise. In the ping terminal we see that h2 is unreachable.

One the set for the project has been deployed and we have checked the correct functioning of the flow, we have decided to create a Maven project in the Controller and proceed to the second half of the work.

Maven is a project manager based on the POM (Project Object Model) concept that allows us to create projects, reports and documents based on certain information.

As mentioned earlier, Opendaylight is a set of OSGi complements, Java classes, other resources and manifests that work over the OSGi framework. Furthermore, it is based on Maven. Therefore, the best thing for our project will be to follow Maven's structure.

The project consists on using Maven to generate a series of directories and files which are compatible with the controller, in such a way that when initiating the controller, the application Java we want to install will be used.

The project is a north-bound Java application installed in the controller, that starts at the same time as the controller. Using the web interface of Opendaylight we should be able to add flows that will filter and redirect the packages according to the protocol established. The flow could also be installed in the switch, as we did in the first part of the project.

The open-source Java application we will use has been developed by the SDNHub Group. We decided to use an already developed app because the objective of our thesis is to teach SDN and not teaching Java programming. In this way, we will adequate the open-source application to our needs, modifying the necessary lines and files in order to make the application work according to our requirements.

Once the application has been installed, we check its correct functioning by testing with different protocols and installing it in different switches using different ways. We use curl commands like in the first part of our work. We use the web interface facilitated by one of the Opendaylight complements (DLux) to insert the flows in a more visual way. We also study the messages that appear in the controller's log at the beginning of the connection, at the end, when a flow is sent, etc.

One of the tests performed consists on deploying a web server in one of the hosts and make a petition from the other. If we configure the flow to filter the HTTP protocol, we will see how packages received in the installed flow increase.

Summing up, the lab test has two parts: a simple and a complex one. In the first one we explain the functioning of the REST API and we generate a configuration file to

discard the packages that go from one host to another. In the second, we have a Java application, installed in the OSGi framework of the controller, and a series of configuring files JSON, pom and manifest that are slightly more complicated and long than those used in the first part.

The first part is a simple and visual flow that states the foundations to understand the functioning of SDN and how the controller and the topologies displayed in Mininet interact. However, the second part is slightly more complex and it is used to study and deeply understand SDN and its potential by using the project manager used by Opendaylight, learning to install an application in the controller, how to execute it and how flows can be added, modified or deleted from the web interface facilitated by the DLux feature.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este capítulo se van a describir los principales aspectos, motivaciones y objetivos de este Trabajo Fin de Grado, así como la estructura del documento.

1.1. Descripción y motivación

El auge de los dispositivos móviles y de las aplicaciones, así como los servicios en la nube, son tendencias que están llevando a las empresas a reexaminar las arquitecturas de red tradicionales y evaluar soluciones que permitan mayor agilidad utilizando herramientas basadas en software y virtualización. Por ello en los últimos años las tendencias en el ámbito de las Telecomunicaciones han sido el estudio de SDN y NFV.

Los operadores se han dado cuenta de que las arquitecturas de redes tradicionales no están optimizadas para los requerimientos actuales, tanto de empresas como de usuarios finales.

NFV permite diseñar las redes y dimensionarlas de tal forma que los servicios alojados en servidores y sistemas de almacenamiento estándar, puedan ubicarse en centros de datos, nodos de red o en el propio cliente. La automatización de la gestión y provisión de la red, a través de SDN, es la siguiente fase en la virtualización de la infraestructura de tecnologías de la información y las telecomunicaciones [1].

De este modo, SDN crea una red inteligente, abierta, flexible, escalable y programable, permitiendo que las aplicaciones modifiquen dinámicamente los servicios proporcionados por la infraestructura de red. Por tanto SDN permite una mayor fiabilidad, rendimiento, escalabilidad y optimización del CAPEX (*Capital Expenditure*) y el OPEX (*Operational Expenditure*).

Algunos de estos cambios que introduce SDN, más en detalle, son: [2]

- **Punto único de control de la red**: permite administrar la red con un control más eficiente sobre la infraestructura y automatizando procesos que antes eran manuales.

- **Mayor agilidad**: Con la automatización a través de SDN, los equipos IT de las empresas tienen un nuevo recurso que agiliza su trabajo.

- **Reducción de costes (Capex/Opex)**: Los beneficios obtenidos a partir de SDN permite una reducción de gastos debido a la optimización de recursos y simplificación de las operaciones.

- **Programable**: al poder programar nuevos flujos sobre los equipos de la red, se agilizan los procesos y las empresas consiguen garantizar calidad de servicios.

Todas estas ventajas y la importancia de SDN en el mercado de las telecomunicaciones se ve reflejada en los datos de mercado, con más de 250 millones de dólares de capital de riesgo en *startups* y más de 1.5 billones de dólares en adquisiciones relacionadas con esta arquitectura de red en el mundo [2].

Estas características que nos proporciona SDN han motivado su estudio y por ende el desarrollo de este Trabajo Fin de Grado.

Debido a la necesidad de exportar el trabajo de forma funcional en un futuro decidimos realizar su desarrollo en un solo equipo y trabajando sobre máquinas virtuales. De este modo nos garantizamos que funcionará en los PCs del laboratorio una vez sea necesario realizar las prácticas.

Para ello hemos utilizado la herramienta de virtualización Virtual Box, de esta forma utilizamos una máquina virtual corriendo Linux y el controlador elegido y en otra la herramienta de virtualización de redes Mininet.

Dicha herramienta se encarga de crear redes virtuales, con un *kernel* real, *switches* y aplicaciones en una sola máquina virtual por medio de un comando, haciéndolo muy útil como método de estudio y desarrollo [3].

A pesar de que este despliegue se podría haber llevado a cabo sobre diferentes máquinas virtuales con una función cada una, hosts, *switches* y controlador, decidimos investigar y probar Mininet para saber si podría utilizarse como herramienta de virtualización para llevar a cabo diferentes tipos de prácticas en la Universidad.

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Grado consiste en el diseño de un guion de prácticas que pueda ser utilizado por la universidad para la enseñanza de SDN. Para la consecución del objetivo principal, se ha propuesto diseñar y desplegar una red definida por software sobre máquinas virtuales. Para ello se van a estudiar y poner en práctica los siguientes componentes en los que se basa SDN:

- Herramientas de virtualización de red: Mininet
- Protocolo OpenFlow
- Controladores: OpenDaylight

Una vez estudiados y probados estos elementos decidimos utilizar una aplicación en Java para mostrar más a fondo el funcionamiento de SDN y alguno de sus posibles usos.

1.3. Estructura de la memoria

En este apartado se muestra una versión esquematizada del contenido total del documento, para facilitar su lectura.

Capítulo 1. Introducción: acercamiento general al Trabajo Fin de Grado y motivaciones y objetivos del mismo.

Capítulo 2. Introduction: acercamiento general al Trabajo Fin de Grado, y motivaciones y objetivos del mismo (versión inglesa).

Capítulo 3. Estado del arte: descripción detallada de SDN, OpenFlow, OpenDaylight y Mininet.

Capítulo 4. Diseño y pruebas: descripción y esquema del despliegue de máquinas virtuales y la aplicación, así como las pruebas realizadas.

Capítulo 5. Guion de prácticas: desarrollo guiado de la memoria de prácticas que se utilizará en la universidad.

Capítulo 6. Planificación y presupuesto: planificación del TFG y presupuesto.

Capítulo 7. Conclusiones y trabajos futuros: exposición de las conclusiones obtenidas y posibles mejoras sobre el Trabajo Fin de Grado.

Capítulo 8. Conclusions and future Works: exposición de las conclusiones obtenidas y posibles mejoras sobre el Trabajo Fin de Grado (versión inglesa).

Anexos: detalla la planificación, el marco regulador y algunos puntos adicionales.

2. INTRODUCTION

In this chapter we will describe the main aspects, motivations and objectives of this Final Bachelor Thesis (TFG) as well as the document's structure.

2.1. Description and motivation

The increasing usage of mobile devices, applications and cloud-based services are making companies reevaluate their traditional network architecture structures. The evolving scene is leading them to consider solutions that allow an increasing speed from the use of software-based tools and virtualization. For this reason, the latest trends in the area of telecommunications are based on the study of SDN and NFV.

The operators have realized that traditional network architectures are not optimized for current requirement both from companies and final users.

NFV allows the design and dimension of networks in a way that services hosted in servers and standard storage systems can be located in data hubs, network nodes or in the client itself. The automatization of network provision and management using SDN constitutes the next phase in the virtualization of the ITC structure.

This way, SDN creates a smart, open, flexible, scalable and programmable network allowing apps to dynamically modify the services provided by the network's infrastructure. Therefore, SDN improves CAPEX (Capital Expenditure) and OPEX (Operational Expenditure) by adding the following attributes: trust, efficiency, scalability and optimization.

Some of the changes introduced by SDN, explained in further detail are:

- Unique point for network control: SDN allows to administrate the network using a more efficient control over the infrastructure and automating former manual processes.
- Increased speed: with SDN-based automation, the company's IT devices have an extra resource to speed up their jobs
- Cost reduction (Capex/Opex): benefits obtained from SDN reduce costs due to resource optimization and operations' simplification
- Programmable: by programming new flows on network devices, processes are accelerated and companies are able to guarantee the quality of their services

Market data reflects both the advantages and the importance of SDN in the IT sector. SDN- related startups have received more than $250 million dollars investment and more than $1.5 billion have been spent in acquisitions related to this network architecture worldwide.

All these SDN-related facts and improvements have sparked our interest on the subject, and therefore motivated its study and the subsequent elaboration of this Bachelor Thesis.

The future needs to export this work in a functional way, have made us decide to develop the project on a single device, working with several virtual machines. By using this method, we guarantee the functioning in the Lab's PC once trials need to be made. In order to accomplish our purpose, we have used Virtual Box. By using this virtualization tool we were able to work with a virtual machine running Linux and the selected controller, and another machine running the network virtualization tool Mininet.

Mininet is a useful method for the study and development of our project as it creates virtual networks with a real kernel, switches and applications in a single virtual machine by using just a command.

Even if this display could have been made using other virtual machines with a function on each (hosts, switches and controller), we decided to research and test if Mininet could be used as a virtualization tool for trials in the University.

2.2. Objectives

The main objective of this Final Bachelor Thesis (TFG) is to design and display a software-defined network that can be used as a lab test in University. For the attainment of the aforementioned goal we are going to study and test the following concepts on which SDN is based:

- Network Virtualization Tools: Mininet
- OpenFlow Protocol
- Controllers: OpenDaylight

After these elements have been studied and tested, we have decided to use a Java application to get a deeper vision on the functioning of SND and some of its possible uses.

2.3. Project Structure

To simplify the reading of this document, we proceed to depict a summarized version of the contents of the project.

Chapter 1. Introduction (Spanish): general view of the thesis, motivations and objectives.

Chapter 2. Introduction (English)

Chapter 3. State of the art: detailed description of SDN, OpenFlow, OpenDaylight and Mininet.

Chapter 4. Lab test design: description and display map of virtual machines and the application

Chapter 5. Test guide: guided development of the test report to be used in the University

Chapter 6. Planning and budget.

Chapter 7. Conclusion and future work (Spanish): statement of the project findings and possible improvements to be made

Chapter 8. Conclusion and future work (English)

Annexes: detailed planning, regulations and budget

3. ESTADO DEL ARTE

En este capítulo haremos una descripción sobre redes definidas por software (SDN), el protocolo Openflow, el controlador Opendaylight y finalmente sobre la herramienta de virtualización de redes Mininet.

3.1. Redes definidas por software (SDN)

SDN es el acrónimo de Software Defined Networking y su principal objetivo es la implementación en software de ciertas funciones de red. Para entender correctamente el funcionamiento de SDN primero debemos explicar las funciones básicas de un *router* o un *switch* en una red IP. Las dos áreas de funcionamiento principales son:

1. **Plano de control**: genera una tabla de rutas para saber por qué interfaz física y a qué destino debe enviar un paquete. Para ello puede utilizar tanto rutas estáticas pre configuradas, como rutas dinámicas aprendidas por protocolos de enrutamiento. Todas las rutas son almacenadas en la tabla RIB (*Routing Information Base*).

2. **Plano de datos/reenvío**: se encarga de reenviar un paquete de la interfaz de entrada del *router* al siguiente salto de la red, utilizando la lógica aplicada por el plano de control.

Estas dos funcionalidades, en las redes tradicionales, se llevaban a cabo en todos los *router* y *switches* de la red.

Lo que SDN plantea es separar dichas funciones e implementar en software y de forma centralizada la función de control.

Por tanto los tres puntos importantes de SDN son:

1. Separación del plano de control y el plano de datos.

2. Centralización de la función de control.

3. Implementación en software del plano de control.

Centralizar la función de control e implementarla en software permite la programación de la red y el despliegue de funcionalidades de forma ágil y flexible. A continuación se muestra una imagen donde se ve claramente la diferencia entre ambas arquitecturas:

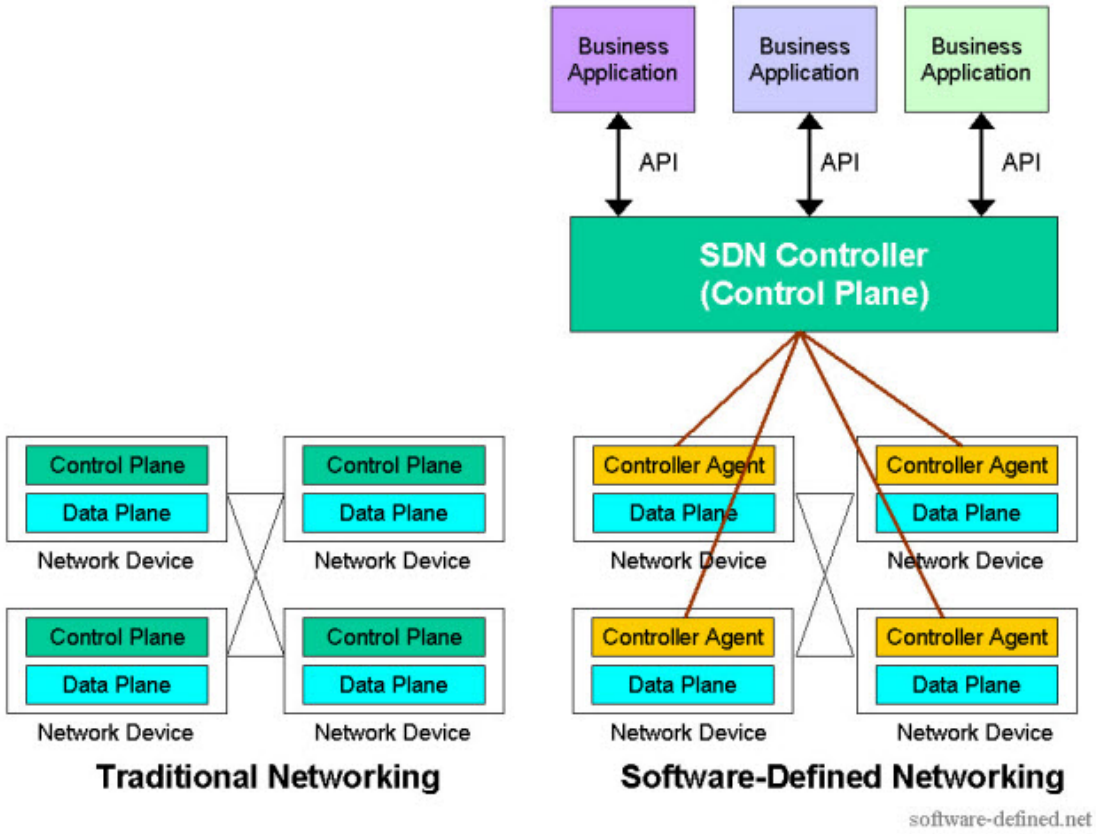


Figura 1: Tipos de Networking.

Fuente: http://www.software-defined.net/

En la parte izquierda de la imagen vemos la arquitectura tradicional de red y podemos ver cómo tanto el plano de control como el plano de datos se encuentran en el mismo dispositivo de la red. Si vamos a la parte derecha vemos la arquitectura que SDN nos plantea, para un mayor entendimiento vamos a explicar ciertos puntos: [4]

- **Aplicaciones del negocio (*Business Application*)**: Aplicaciones finales utilizables por el usuario: video conferencia, gestión de relaciones con clientes y gestión de la cadena de suministro.

- **Controlador SDN**: encargado del plano de control con todas las funciones de un *switch* tradicional.

- **Northbound API**: encargada de la comunicación entre la capa de control y las aplicaciones empresariales.

- **Southbound API**: permiten las comunicaciones entre el plano de control y la infraestructura. Los protocolos que permiten esta comunicación son XMPP, el protocolo de configuración de red y Openflow, del cual hablaremos más adelante.

La necesidad de crear esta nueva arquitectura de red viene dada por el auge de la virtualización de redes y los servicios en la nube así como las siguientes claves: [1]

- **Cambio en los patrones de tráfico**: Al contrario que en las tradicionales aplicaciones cliente-servidor en las que sólo entraban en juego un cliente y un servidor, las aplicaciones actuales interaccionan entre varias bases de datos y servidores, requiriendo que el tráfico vaya de máquina en máquina hasta que los datos son devueltos al usuario.

- **Consumismo de las tecnologías de la información**: Cada vez más los usuarios utilizan sus dispositivos personales, *smartphones*, *tablets* y portátiles, para acceder a las redes empresariales, haciendo que los empleados de IT tengan que permitir su uso a la vez que protegen la propiedad intelectual y los datos de la empresa.

- **Servicios en la nube**: Las empresas han adoptado los servicios en la nube tanto públicos como privados y quieren que el acceso a los recursos se realice de forma ágil. Además todo esto debe llevarse a cabo en un entorno seguro y sensible a modificaciones de almacenamiento, computacionales y de recursos de red.

- **Big Data**: El crecimiento de Big Data y las grandes bases de datos, requiere de un aumento de las capacidades de red en los centros de datos. Por tanto los operadores de red se enfrentan a la difícil tarea de aumentar la red evitando que sufra cualquier tipo de pérdida de conectividad.

3.1.1. SDN y NFV

La virtualización de las funciones de red (NFV) se encarga de sustituir los componentes físicos de la red como *routers*, *switches* o *firewalls* por software ejecutado en servidores, incrementando el ritmo de innovación en los servicios de red.

Es decir, NFV se encarga de abstraer el hardware en máquinas virtuales implementadas en software con los requisitos necesarios para su buen funcionamiento.

De esta forma las aplicaciones tienen una máquina donde funcionar independientemente del hardware que las soporta.

A pesar de que SDN y NFV sean conceptos independientes y la virtualización no requiera necesariamente de SDN, ambos conceptos combinados serán más eficientes y se beneficiarán mutuamente logrando el máximo rendimiento de la infraestructura.

De esta manera SDN se encargaría de controlar el tráfico de la red y NFV se encargaría de asignar los servicios de red y seguridad a las máquinas virtuales de acuerdo a sus necesidades y de forma persistente, es decir, si esa máquina virtual se mueve de un host a otro, los servicios se mantendrían vinculados.

3.1.2. Beneficios de SDN

A pesar de haber nombrado algunas de las ventajas que nos aporta SDN y algunas aplicaciones que podría tener, ¿qué beneficios tiene SDN?.

A continuación listamos algunos de los más importantes: [5]

- **Red programable**: la red es directamente programable gracias a la separación del plano de datos y el plano de control.

- **Control centralizado**: toda la inteligencia de la red se encuentra centralizada en el controlador SDN, permitiendo una visión global de la red.

- **Reducción del CapEx y el OpEx**: SDN reduce el uso y la necesidad de hardware, así como el coste de operaciones, gracias a algoritmos que permiten el control de la red y de los elementos de la red, facilitando el desarrollo, el control, el escalado y la automatización de la red.

- **Agilidad y flexibilidad**: SDN ayuda a las empresas desplegar rápidamente aplicaciones, servicios e infraestructura.

- **Innovación**: software defined networking permite a las compañías crear nuevos tipos de aplicaciones, servicios y modelos de negocio que les den mayor valor.

3.1.3. Usos y aplicaciones de SDN

Software defined networking tiene variedad de aplicaciones en los entornos de red. Separar el plano de control del plano de datos permite tener redes programables, simplificando el desarrollo y despliegue de nuevos servicios de red y protocolos.

Algunos de los entornos donde podría implantarse SDN son: [6]

- **Redes empresariales**: Habitualmente las empresas tienen grandes despliegues de red y un control adecuado es muy importante. Las redes SDN pueden ser utilizadas para manejar las políticas y la monitorización de la red. Además permiten simplificar la red implementando funcionalidades como *firewalls*, NAT, balanceadores de carga… en el controlador.

- **Data centers**: Los data centers han evolucionado en los últimos años tratando de satisfacer el alto y rápido crecimiento de la demanda. El control del tráfico y las políticas son críticas al trabajar a gran escala, especialmente cuando existe la parada de algún servicio que puede disminuir la productividad. Otro problema, es el alto consumo energético, lo cual es capaz de solucionar SDN mejorando la eficiencia a través de métodos que apagan o encienden *switches* en función de las necesidades de la red, permitiendo un ahorro del 25-62% de la energía.

- **Infraestructuras de red de acceso inalámbrico**: Hay un gran interés por parte de los operadores móviles en aplicar SDN debido a que simplificaría la administración de sus redes y permitirían nuevos servicios que soporten el crecimiento exponencial del tráfico previsto para las redes LTE.

- **Redes ópticas**: Manejar el tráfico de datos como flujos, permite a las *software defined networks* y a las redes Openflow en particular, soportar e integrar múltiples tecnologías de red. De acuerdo al Optical Transport Working Group (OTWG) creado en 2013 por la Open Network Foundation (ONF), los beneficios en las redes ópticas de transporte aplicando SDN y el protocolo Openflow serían: mejorar el control de las redes ópticas y su flexibilidad, permitiendo el despliegue de sistemas de control de terceros y nuevos servicios de virtualización.

- **Hogares y pequeñas empresas**: Varios proyectos han estudiado cono aplicar SDN en pequeñas redes como las de los hogares o las pequeñas empresas. Esto se debe al reciente crecimiento de dispositivos de red de bajo coste, su consecuente aumento de complejidad y a la necesidad de tener mayor control y seguridad en estas redes.

3.2. Openflow

Lo que OpenFlow [7] propone es una forma para que los investigadores puedan experimentar con protocolos en las redes que utilizan a diario. OpenFlow está basado en un Ethernet *switch* con una tabla de flujos interna y una interfaz estandarizada para añadir y eliminar dichos flujos.

El objetivo principal de OpenFlow es permitir a los investigadores experimentar en *switches* heterogéneos de manera uniforme y con una alta densidad de puertos. Por otro lado los vendedores no tendrían que exponer el funcionamiento interno de sus *switches*. Además permite a los investigadores probar sus ideas en entornos de trabajo reales y a gran escala.

3.2.1. Protocolo OpenFlow

La idea principal es explotar el hecho de que la mayoría de los Ethernet *switches* y *routers* modernos tienen tablas de flujos que van a velocidad de línea e implementan firewalls, NAT, QoS y recolectan estadísticas. A pesar de que la mayoría de las soluciones propietarias tienen diferencias, existen algunos puntos comunes entre todas, y eso es lo que trata de aprovechar OpenFlow [7].

OpenFlow proporciona un protocolo para programar las tablas de flujo de los *switches* y *routers* de forma dinámica. De este modo los investigadores pueden probar nuevos protocolos de enrutamiento, modelos de seguridad, esquemas de red e incluso alternativas a IP, mientras que el plano de datos se encontraría aislado y seguiría procesando de la forma tradicional.

Las tres características principales de un *router* OpenFlow son las siguientes:

- **Tabla de flujos**: cada flujo lleva asociado una acción.

- **Canal seguro**: se encarga de conectar el *switch* con un controlador, permitiendo comandos y paquetes entre ellos.

- **Protocolo OpenFlow**: proporciona al controlador una forma abierta y estandarizada para comunicarse con el *switch*.

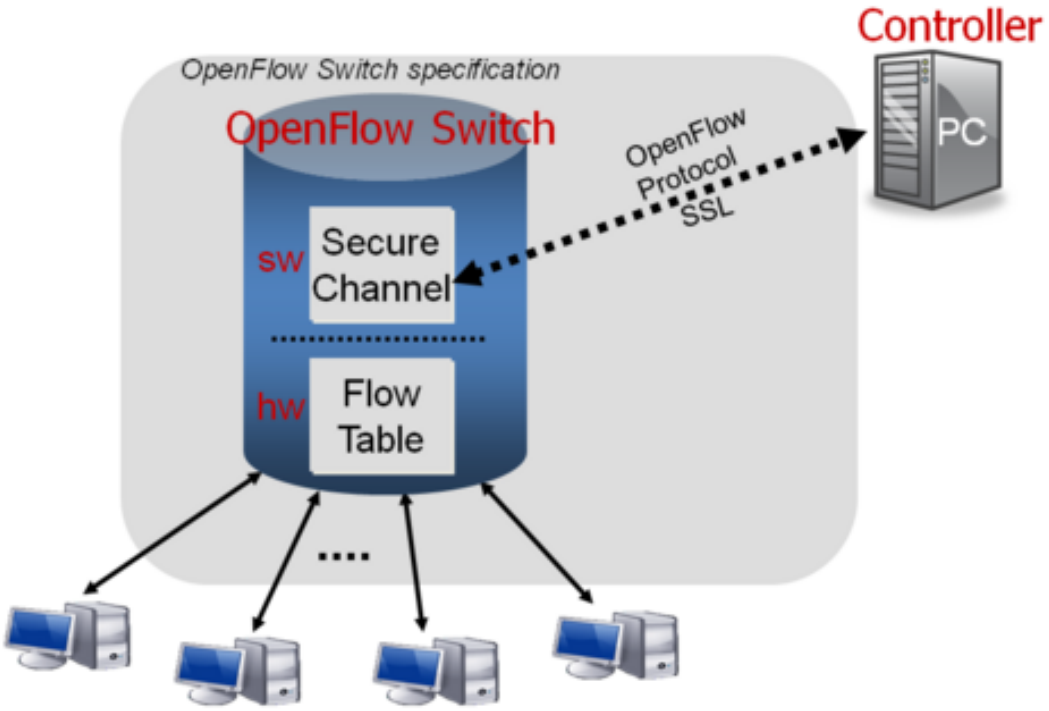


Figura 2: Especificaciones de un switch OpenFlow

Fuente: McKeown et al. (2008)

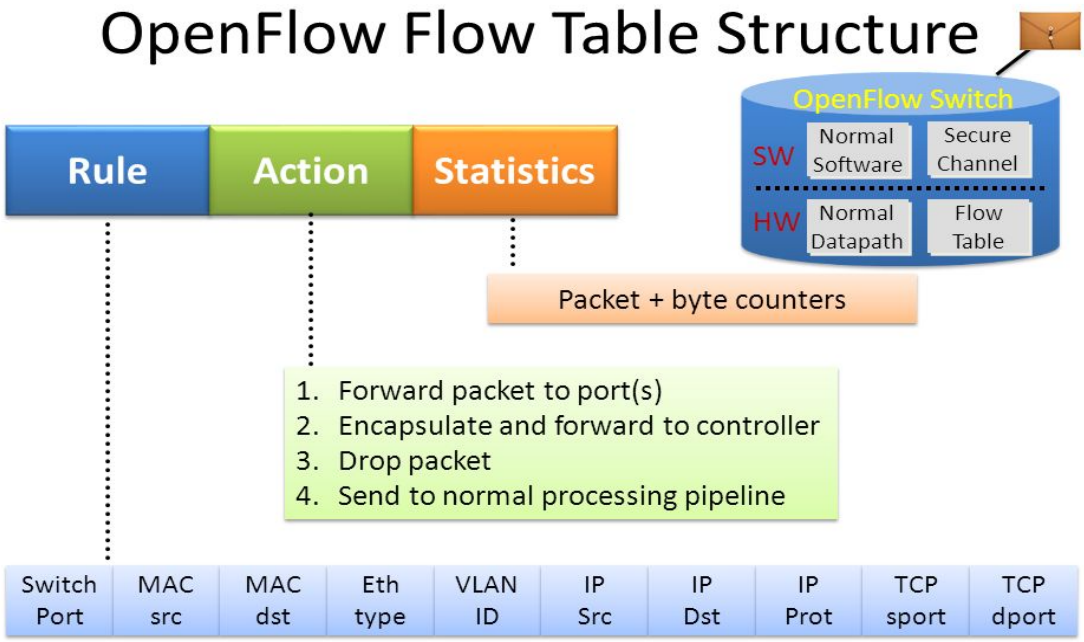


Figura 3: Estructura de Tabla en un switch OpenFlow

Fuente: McKeown et al (2013) [8]

Utilizando el protocolo OpenFlow el controlador puede añadir, actualizar y borrar flujos de las tablas, de forma reactiva o proactiva. Existen una gran cantidad de parámetros que pueden utilizarse para definir un flujo, algunos de ellos son los puertos por los que se reciben los paquetes, el puerto origen, el puerto Ethernet destino, la etiqueta VLAN y otras características de los paquetes.

Al realizar la búsqueda en las tablas de flujo para encontrar coincidencias con los paquetes entrantes, si existe coincidencia se realizará la acción asociada a dicho flujo, si no existe ningún entrada en la tabla de flujos asociada a dicho paquete, el controlador deberá crear un nuevo flujo y se lo enviará al *switch* para que pueda procesar ese paquete, se buscarán coincidencias en la siguiente tabla de flujos o finalmente será descartado. Todo ello dependerá de cómo este hecha la configuración del procesado de paquetes si no se encuentran coincidencias.

3.2.2. Puertos OpenFlow

Los puertos OpenFlow [9] son las interfaces de red que transmiten los paquetes entre el procesado del protocolo y el resto de la red. El número de puertos OpenFlow configurados por el *switch* no tiene por qué coincidir con el número de interfaces de red físicas del propio *switch*.

Un *switch* OpenFlow debe soportar los siguientes tipos de puertos:

- **Puertos estándar**: pueden ser utilizados como puertos de entrada y de salida, se pueden utilizar en grupos y tienen contadores de puerto.

- **Puertos físicos**: son definidos por el *switch* y se corresponden con las interfaces físicas del *switch*.

- **Puertos lógicos**: no se corresponden directamente con las interfaces físicas del *switch*, pueden estar mapeados a diferentes puertos hardware.

- **Puertos reservados**: están definidos por la especificación del protocolo y tienen funciones específicas. Los más importantes son:

  → **All**: representa todos los puertos del *switch*. Solo puede ser usado como puerto de salida y se utiliza para enviar paquetes específicos.

→ **Controller**: representa el canal de comunicación con el controlador. Puede ser usado como puerto de entrada y de salida.

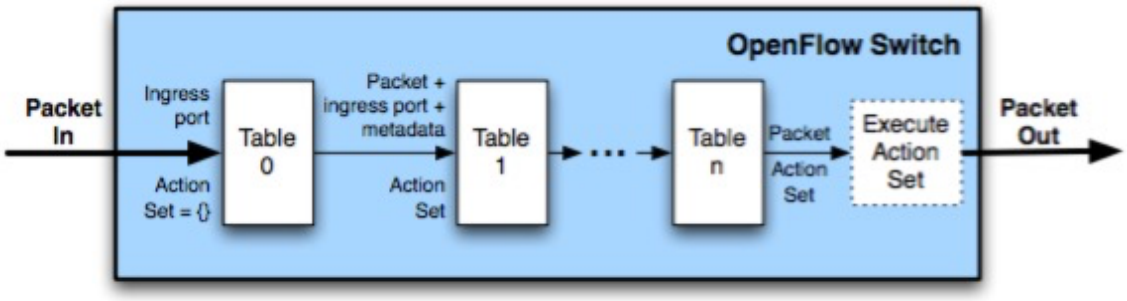
→ **Table**: representa el comienzo de la de las tablas de flujos.

→ **In_Port**: representa el puerto de entrada de los paquetes.

→ **Any**: puerto especial utilizado por algunos comandos OpenFlow cuando no se especifica el puerto.

3.2.3. Tablas OpenFlow

Las tablas de flujos [9] en un *switch* OpenFlow están secuencialmente numeradas, empezando por el cero. El procesado de los paquetes siempre empieza en la primera tabla, primero se buscan coincidencias con las entradas de flujos de la tabla cero; las siguientes tablas serán utilizadas en función de la salida de la tabla anterior.



(a) Packets are matched against multiple tables in the pipeline

Figura 4: Procesamiento de paquetes switch OpenFlow

Fuente: ONF (Junio 2012)

Las tablas de flujos están compuestas por entradas de flujo con los siguientes campos:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: Main components of a flow entry in a flow table.

Figura 5 Principales componentes de una entrada de flujo

Fuente: ONF (Junio 2012)

1) **Match fields**: para buscar coincidencias con los paquetes.
2) **Priority**: prioriza los flujos
3) **Counters**: actualiza los el número de paquetes coincidentes.
4) **Instructions**: modifica la acción a realizar.
5) **Timeouts**: máximo tiempo de vida del flujo.
6) **Cookie**: dato elegido por el controlador y utilizado para filtrar flujos modificados, borrados o estadísticas.

El diagrama de flujo de un paquete en un *switch* OpenFlow es el siguiente:

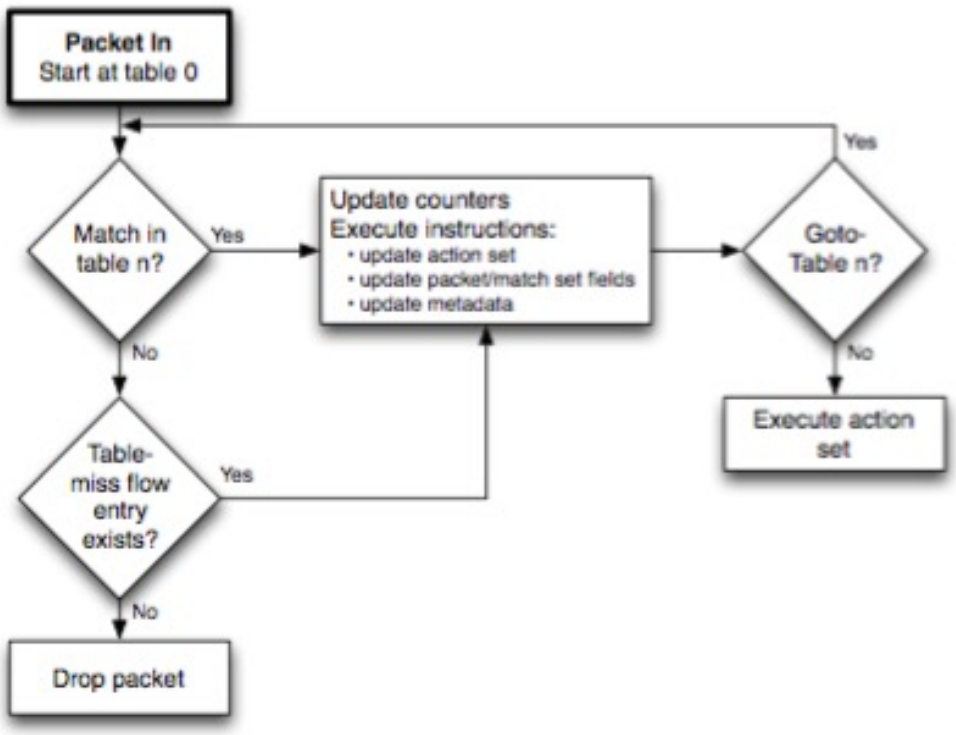


Figura 6: Diagrama de flujo del procesado de paquetes en un switch OpenFlow

Fuente: ONF (Junio 2012)

3.2.4. Contadores

Los contadores [9] se mantienen por cada tabla de flujo, entrada de flujo, puertos y cola. Pueden estar implementados en software y ser mantenidos por contadores hardware con rangos más limitados.

3.2.5. Instrucciones

Cada entrada de flujo contiene un conjunto de instrucciones [9] que son ejecutadas cuando un paquete coincide con las características del flujo. Estas instrucciones realizan diferentes acciones sobre el paquete:

Un *switch* no tiene por qué soportar todos los tipos de instrucciones, sólo algunas son requeridas:

1) **Write-Actions action(s)**: si la acción existe, se sobrescribe, si no, se añade.
2) **Goto-Table next-table-id**: Indica la siguiente tabla del procesado. Las entradas de flujo de la última tabla no pueden incluir esta instrucción.

El conjunto de instrucciones asociadas a un flujo solo pueden tener una instrucción de cada tipo y siempre se mantiene un orden de ejecución.

Un *switch* puede rechazar un flujo si no es capaz de ejecutar sus instrucciones.

Las acciones requeridas que debe soportar un *switch* OpenFlow son: Output, que se encargará de reenviar un paquete a un puerto OpenFlow concreto, *Drop*, descartará paquetes que no tengan asociados alguna acción y *Group*, procesará un paquete en función del tipo de grupo al que pertenezca.

3.2.6. Canal OpenFlow

El canal OpenFlow [9] es la interfaz encargada de conectar cada *switch* OpenFlow al controlador. A través de esta interfaz el controlador se encarga de configurar y controlar los *switches*, recibiendo eventos del *switch* y enviando paquetes hacia él.

Existen tres tipos de comunicación entre el controlador y el *switch*:

1) **Controller-to-*Switch***: son mensajes iniciados por el controlador y que pueden, o no, requerir respuesta del *switch*: petición de características al *switch*, configuración de parámetros en el *switch*, modificación de estado, etc.

2) **Asynchronous**: son enviados por el *switch* hacia el controlador para avisar de la llegada de un paquete, el estado del *switch* o un error. Los cuatro tipos de mensajes asíncronos son: *Packet-in*, *Flow-Removed*, *Port-status* y Error.

El primero se utiliza para transferir el control de un paquete al controlador, *Flow-Removed*, informa al controlador de la eliminación de un flujo, *Port-Status*, notifica al controlador de los cambios en los puertos y el mensaje *Error* se encarga de notificar de algún problema.

3) **Symmetric**: se envían en ambas direcciones sin solicitud previa, son: *Hello*, *Echo* y *Experimenter*. El mensaje *Hello* es intercambiado cuando se inicia la conexión, *Echo* se utiliza para verificar el estado de la conexión y *Experimenter* para ofrecer funciones adicionales.

3.3. Opendaylight

En este punto vamos a hablar del controlador que hemos utilizado en este Trabajo Fin de Grado.

Un controlador se encarga de añadir y eliminar flujos en las tablas como hemos especificado anteriormente. Hay diversos controladores, pero hemos decido utilizar Opendaylight, debido a la mayor cantidad de documentación que hará más sencilla la tarea de desplegar el escenario.

Opendaylight [10] es un proyecto *open-source* promovido por la Linux Foundation, una asociación de empresas que desarrollan proyectos en conjunto.

Este controlador está implementado en software contenido dentro de su propia máquina virtual de Java, por tanto puede ser desplegado en cualquier plataforma que soporte Java. Al ser *open-source* contiene APIs abiertas que pueden ser utilizadas por los desarrolladores para aplicaciones. Opendaylight soporta el framework OSGi y REST bidireccional.

El framework OSGi se utiliza para aplicaciones que se ejecutan de modo reactivo mientras que las desarrolladas mediante REST API carecen de esta posibilidad. Las aplicaciones utilizan el controlador para reunir información de la red y ejecutar algoritmos para realizar análisis y después utilizar el controlador Opendaylight para crear nuevas reglas para aplicar a la red [11].

Opendaylight es un compendio de *bundles* OSGi, clases Java y otra seria de recursos que son enlazables para realizar diversas tareas. Algunos de estos servicios nos permiten realizar tareas de red como ver que dispositivos se encuentran en ella, recopilación de estadísticas, etc. Estos *bundles* se pueden instalar y desinstalar de forma sencilla, de tal modo que permite al desarrollador elegir lo que quiere utilizar en cada momento.

Si nos fijamos en la figura 7 vemos que la interfaz de bajo nivel (Southbound Interfaces & Protocol Plugins) soporta multitud de protocolos. Estos módulos están conectados a la Service Abstraction Layer (SAL), cuya función es "traducir" las órdenes o servicios requeridos sin importar el dispositivo de red que haya por debajo. Esto hace que Opendaylight sea más versátil que otros controladores, ya que permite utilizar diferentes equipos, protocolos, etc.

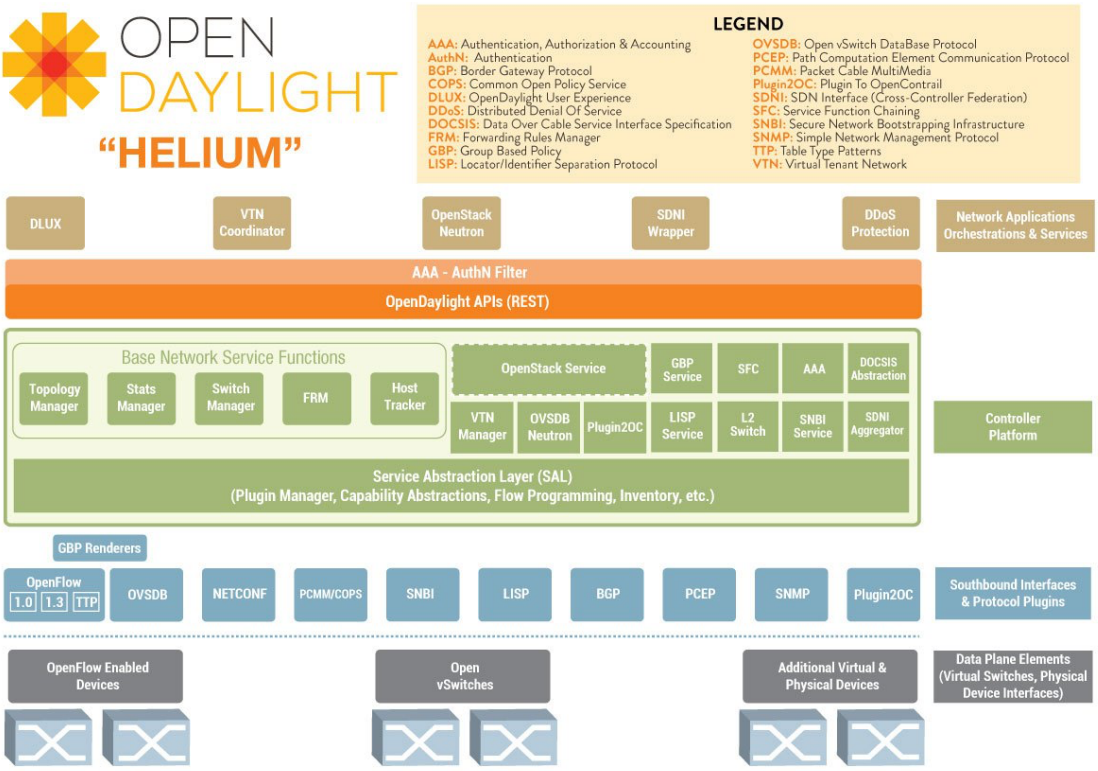


Figura 7: Estructura del controlador Opendaylight

Fuente: Opendaylight

Por tanto la SAL se encarga de conectar los *plugins* del protocolo con los Módulos de Función de Red. Las API SAL son los contratos que establecen los *plugins* con los módulos para poder comunicarse entre ellos, existen dos tipos: AD-SAL y MD-SAL.

En este Trabajo Fin de Grado se ha optado por utilizar MD-SAL ya que AD-SAL está obsoleto. En la figura 8 se muestran las principales diferencias entre AD-SAL y MD-SAL [12]:

AD-SAL	MD-SAL
API-Driven SAL	Model-Driven SAL
The SAL APIs, request routing between consumers and providers, and data adaptations are all statically defined at compile/build time.	The SAL APIs, request routing between consumers and providers are defined from models, and data adaptations are provided by 'internal' adaptation plugins.
The AD-SAL typically has both NB and SB APIs even for functions/services that are mapped 1:1 between SB Plugins and NB Plugins.	The MD-SAL allows both the NB plugins and SB plugins to use the same API generated from a model. One plugin becomes an API (service) provider; the other becomes an API (service) Consumer.
In AD-SAL there is a "dedicated" REST API for each northbound/southbound plugin.	MD-SAL provides a "common" REST API to access data and functions defined in models .
he AD-SAL provides request routing (selects an SB plugin based on service type) and optionally provides service adaptation, if an NB (Service, abstract) API is different from its corresponding SB (protocol) API.	The MD-SAL provides request routing and the infrastructure to support service adaptation, but it does not provide service adaptation itself; service adaptation is provided by plugins.
Request routing is based on plugin type: the SAL knows which node instance is served by which plugin, and when an NB Plugin requests an operation on a given node, the request is routed to the appropriate plugin which then routes the request to the appropriate node.	Request Routing in the MD-SAL is done on both protocol type and node instances, since node instance data is exported from the plugin into the SAL.
AD-SAL is stateless	MD-SAL can store data for models defined by plugins: provider and consumer plugins can exchange data through the MD-SAL storage
Limited to flow-capable device and service models only	Model agnostic. It can support any device and/or service models and is not limited to flow-capable device and service models only
AD-SAL services usually provide both asynchronous and synchronous versions of the same API method.	In MD-SAL, Service model APIs only provide asynchronous APIs, but they return a 'java.concurrent.Future' object, which allows a caller to block until the call is processed and a result object is available. Same API can be used for both synchronous and asynchronous approach. Thus MD-SAL encourages asynchronous approach to application design but does not preclude synchronous applications.

Figura 8: Diferencias entre AD-Sal y MD-Sal

Fuente: SDN Tutorials

3.4.Mininet

Mininet [3] es un emulador de red que crea redes con *hosts* virtuales, *switches*, controladores y enlaces. Los hosts utilizan el software de red de un sistema operativo Linux estándar y los *switches* soportan el protocolo OpenFlow lo que permite gran flexibilidad y personalización de enrutamientos y SDN.

Mininet soporta multitud de tareas que se benefician al tener un despliegue de red completo en un portátil o PC.

Sus principales características son:

- Proporciona un entorno de pruebas sencillo y barato para desarrollar aplicaciones OpenFlow.
- Permite a los desarrolladores trabajar simultáneamente y de forma independiente en la misma topología.
- Soporta pruebas de regresión a nivel de sistema, que son repetibles y empaquetables.
- Permite realizar pruebas de topologías complejas sin necesidad de una red física.
- Incluye un CLI para pruebas de red y *debugging*
- Soporta topologías personalizadas e incluye topologías predeterminadas, además de una API Python para la creación de redes y pruebas.

Las redes creadas en Mininet utilizan código real incluyendo las aplicaciones estándar de red basadas en Unix/Linux así como el kernel y la pila de protocolos de red. Debido a esto el código desarrollado y probado en Mininet para un controlador OpenFlow, un *switch* modificado o un *host*, puede ser desplegado en sistemas reales realizando pocos cambios, esto es importante ya que un diseño de red que funciona en Mininet generalmente puede ser trasladado directamente a *switches* físicos de forma funcional.

3.4.1. Ventajas y desventajas

Mininet tiene una serie de ventajas y desventajas frente a las redes reales y sistemas de red virtualizados completamente.

Ventajas

- Velocidad de carga
- Mayores despliegues de red
- Proporciona mayor ancho de banda
- Instalación sencilla
- No tiene ningún coste asociado y tiene alta disponibilidad
- Reconfigurable
- Rendimiento configurable

Desventajas

- Actualmente las redes desplegadas por Mininet no pueden exceder la CPU o el ancho de banda del servidor.
- Mininet no puede correr sistemas que no sean compatibles con *switches* OpenFlow o aplicaciones no compatibles con Linux.
- Mininet no hace NAT hacía fuera. Por lo que no podrá salir a internet a menos que se configure explícitamente [13].
- Todos los *hosst* creados por Mininet comparten el mismo sistema de archivos y el mismo espacio PID [13].

4. DISEÑO Y PRUEBAS DE LA PRÁCTICA

En este capítulo se explicará y se realizará un esquema del diseño del sistema a desarrollar. Así como una explicación de las pruebas realizadas para el correcto funcionamiento del escenario desplegado

4.1. Introducción

En este apartado de la memoria se describirán las distintas fases realizadas en el desarrollo de este Trabajo Fin de Grado.

Con este proyecto se pretende realizar el diseño y despliegue de una red definida por software, y realizar un manual de prácticas que más adelante será utilizada con fines docentes.

Para poder realizar la práctica en los puestos de trabajo de la universidad, era necesario definir un escenario formado por un único host, en este caso un portátil, que aloja dos máquinas virtuales desplegadas en Virtual Box. Una de ellas alberga un controlador y la otra un software de emulación de entornos de red. Ambas máquinas virtuales deben tener visibilidad entre sí para que sea posible conectar la topología de red creada en el emulador, con el controlador externo. A continuación se muestra gráficamente el escenario desplegado.

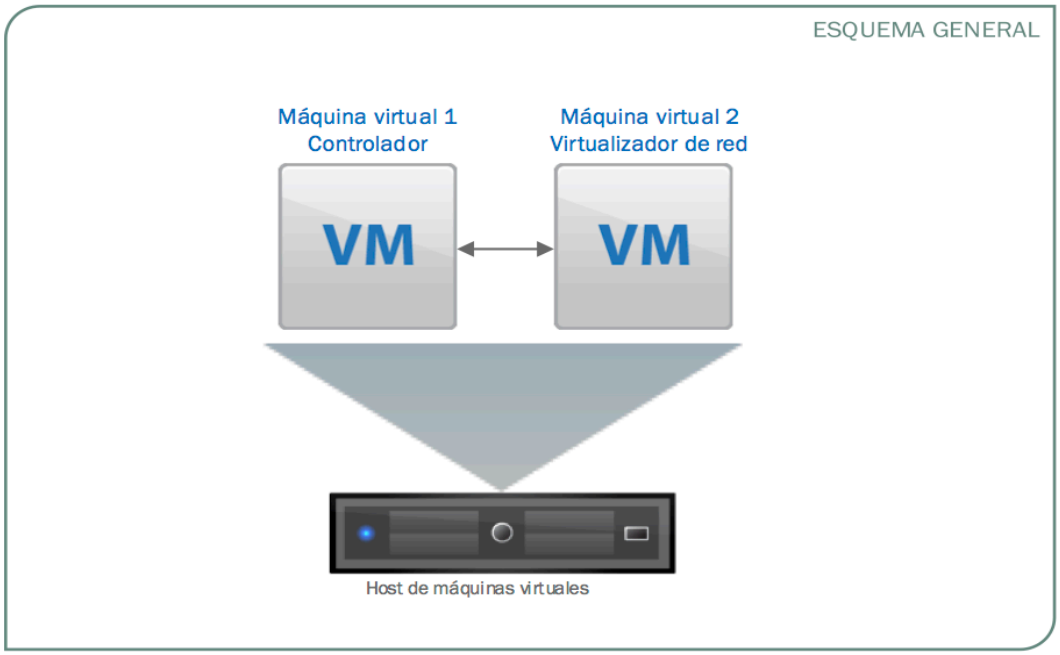


Figura 9: Diseño general del entorno de trabajo

Fuente: Elaboración propia

4.2. Elección del controlador y software de virtualización de redes

Como hemos podido ver en el esquema general del entorno de trabajo, tenemos un controlador y un software de virtualización de redes; en este apartado se explican cuáles y por qué elegimos cada uno de ellos.

4.2.1. Controladores

Actualmente existen multitud de controladores implementados en diferentes lenguajes y preparados para diferentes entornos. Los más destacados son:

- **Ryu** [14]: es un framework de redes definidas por software basado en componentes. Ryu provee componentes software con APIs bien definidas que permiten a los desarrolladores crear nuevas aplicaciones de gestión y control de red. Es compatible con varios protocolos, OpenFlow, Netconf, OF-config,etc. Todo el código esta disponible bajo la licencia de Apache 2.0.

- **Floodlight** [15]: es un controlador de clase empresarial, y está bajo la licencia de Apache. Este controlador está basado en Java y soporta el estándar OpenFlow. Floodlight está diseñado para trabajar con un gran número de *switches*, *routers*, *switches* virtuales y puntos de acceso compatibles con OpenFlow. Es apoyado por una gran comunidad de desarrolladores.

- **NOX/POX** [16]: Nicira Networks desarrollo NOX a la vez que se desarrollaba OpenFlow, y fue el primer controlador SDN basado en este protocolo. En 2008, Nicira donó el desarrollo de NOX a la comunidad de desarrolladores de código libre.

  NOX está desarrollado para aplicaciones de control SDN en C++ y POX es una variante para el desarrollo en Python, actualmente es más utilizado que NOX.

Finalmente, Opendaylight, que ya hemos descrito en el capítulo 2 de este Trabajo Fin de Grado, es el utilizado. Debido a su implementación en Java, la comunidad de desarrolladores que hay detrás del proyecto, su amplia documentación, ejemplos y ser *open-source*, fue el elegido para el desarrollo del TFG.

4.2.2. Software de virtualización de red

El software utilizado como virtualizador de red es Mininet. Elegimos este emulador de red debido a su compatibilidad con OpenFlow, estándar necesario para la

comunicación entre la topología de red y el controlador, y por qué trabaja con Open v*Switch* [17], un sistema de *switch* virtual diseñado para permitir la automatización masiva de la red mientras soporta protocolos estándar.

Por tanto, Mininet reunía todas las características necesarias para el desarrollo del Trabajo Fin de Grado.

4.3. Instalación y prueba de Mininet

Una vez elegido Mininet como software de virtualización de redes, procedemos a su instalación y puesta en marcha. Para ello descargamos de la propia página web [3] la máquina virtual cedida por el equipo Mininet.

Descargada la máquina virtual configuramos en Virtual Box las opciones de red y de rendimiento, de la siguiente forma:

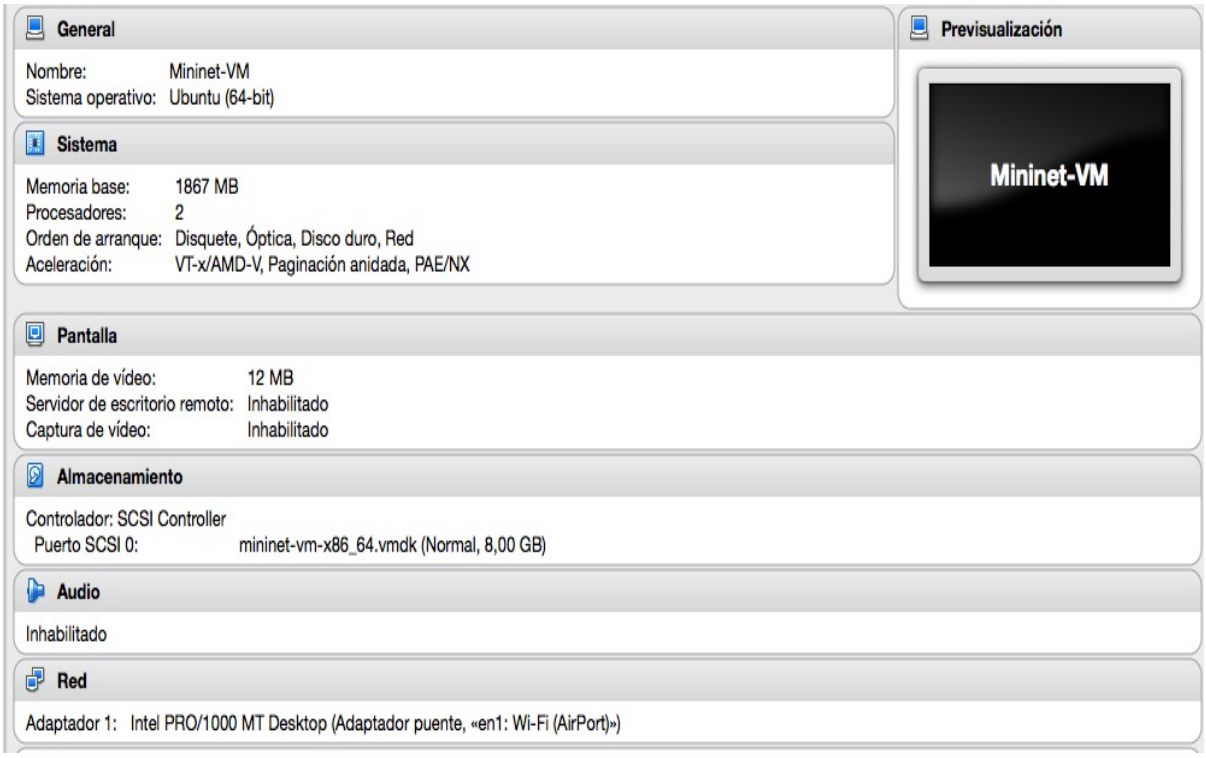


Figura 10: Configuración de la máquina virtual de Mininet

Fuente: Elaboración propia

Lo más importante de esta configuración es configurar el adaptador de red en modo bridge, para poder interconectar el controlador con esta máquina virtual.

A continuación ejecutamos Mininet e introducimos las credenciales para entrar a la máquina virtual:

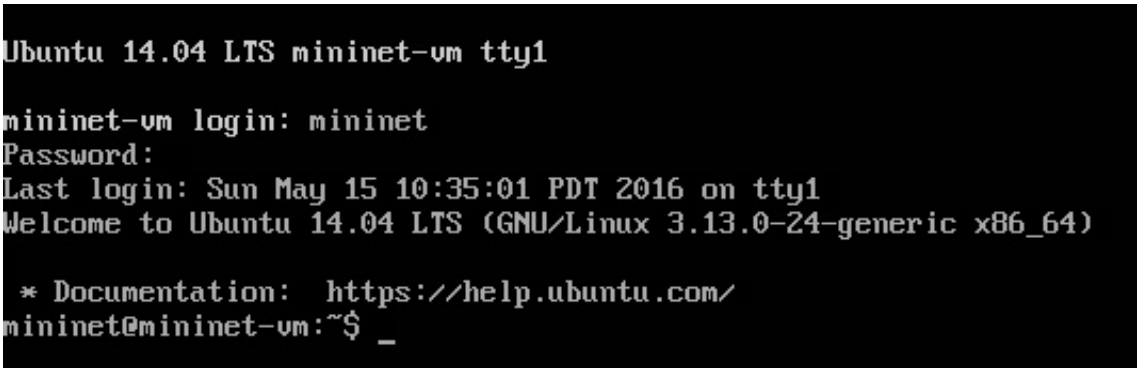


Figura 11: Login en Mininet

Fuente: Elaboración propia

Después de poner en marcha la máquina virtual comenzamos las pruebas de funcionamiento de Mininet. Para ello nos familiarizamos con los comandos CLI de esta herramienta:

1. **Creamos un topología y entramos al CLI de Mininet**:

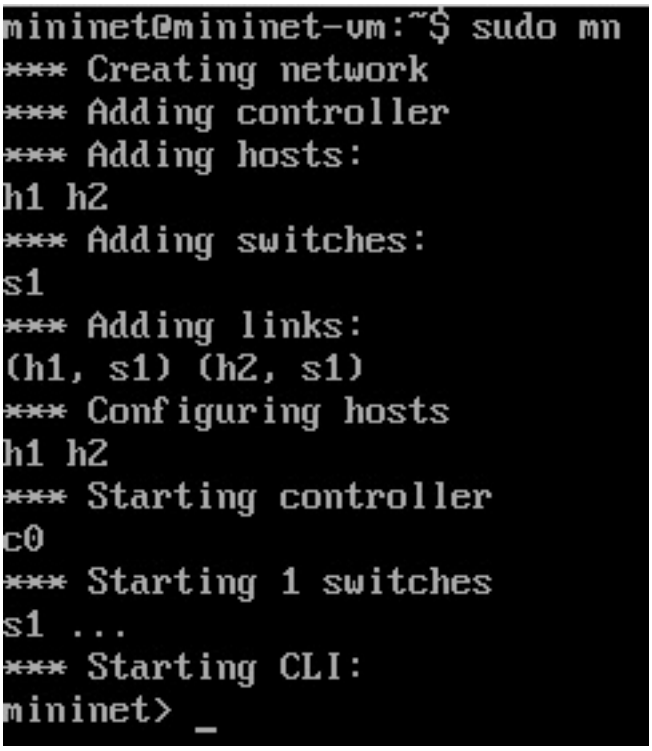


Figura 12: Creación topología "minimal" predeterminada en Mininet

Fuente: Elaboración propia

Al introducir el comando *sudo mn* se genera una topología predeterminada que incluye dos *host* (h1 y h2), un *switch* (s1), un controlador local (c0) y los enlaces entre los *host* el *switch* y el controlador. En la última línea aparece el *prompt* de Mininet.

2. Comandos *nodes, net* y *dump:*

El comando *nodes* muestra los nodos pertenecientes a la topología, el comando *net* los enlaces generados y el comando *dump* información de todos los nodos.

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Figura 13: Comandos "nodes" y "net"

Fuente: Elaboración propia

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1456>
<Host h2: h2-eth0:10.0.0.2 pid=1460>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1465>
<Controller c0: 127.0.0.1:6633 pid=1449>
```

Figura 14: Comando "dump"

Fuente: Elaboración propia

3. Particularización de comandos a un host:

En Mininet si antes de introducir un comando añadimos el nombre de un *host*, por ejemplo h1 vemos cómo el resultado del comando hace referencia única y exclusivamente a dicho *host*, es decir:

```
mininet> h1 ifconfig -a
h1-eth0   Link encap:Ethernet  HWaddr 32:bc:09:10:8c:e7
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 15: Particularización comando "ifconfig" a h1

Fuente: Elaboración propia

En esta imagen vemos como con el comando *h1 ifconfig –a* se muestran las interfaces de red del *host* virtual h1. La interfaz *h1-eth0* no la veremos si realizamos un *ifconfig* sobre el sistema Linux primario, ya que es específica del proceso *host* h1.

4. Conectividad entre hosts:

Para comprobar la conectividad entre *hosts* realizamos un comando *ping* entre h1 y h2.

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=11.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.064/11.064/11.064/0.000 ms
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.68 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.681/1.681/1.681/0.000 ms
```

Figura 16: Prueba de conectividad entre host

Fuente: Elaboración propia

En la captura anterior vemos dos *ping* realizados al mismo *host*. Sin embargo, si nos fijamos en los tiempos de llegada, vemos que el segundo es bastante inferior al primero. Esto se debe al control de tráfico llevado a cabo por OpenFlow.

El primer *host* realiza una consulta ARP para conocer la MAC del segundo *host*, esto genera un mensaje *packet_in* hacia el controlador. El controlador envía un mensaje *packet_out* a *broadcast* haciendo que le llegue al segundo *host*, el cual, una vez ha recibido el *arp_request* responde con un *arp_reply*. La respuesta va al controlador que la envía al *host* 1 e instala un flujo para cubrir los paquetes ICMP (*ping*) en el *switch*, haciendo que los próximos paquetes generados por un *ping* pasen directamente a través del *switch* y no haya control de trafico.

Existe otra manera de comprobar la conectividad de la red virtual y consiste en realizar un comando *pingall*.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figura 17: Comando "pingall"

Fuente: Elaboración propia

4.4.Instalación y pruebas de Opendaylight

Conocido el funcionamiento de Mininet, comandos básicos y su potencial, instalamos la máquina virtual que aloja el controlador Opendaylight. En esta máquina utilizamos el sistema operativo Lubuntu 14.04.3, elegimos este sistema operativo por ser más ligero que Ubuntu y poder, como dijimos en puntos anteriores, desplegar todo el escenario del Trabajo Fin de Grado en un solo *host*. A continuación se muestra la configuración de la máquina virtual:

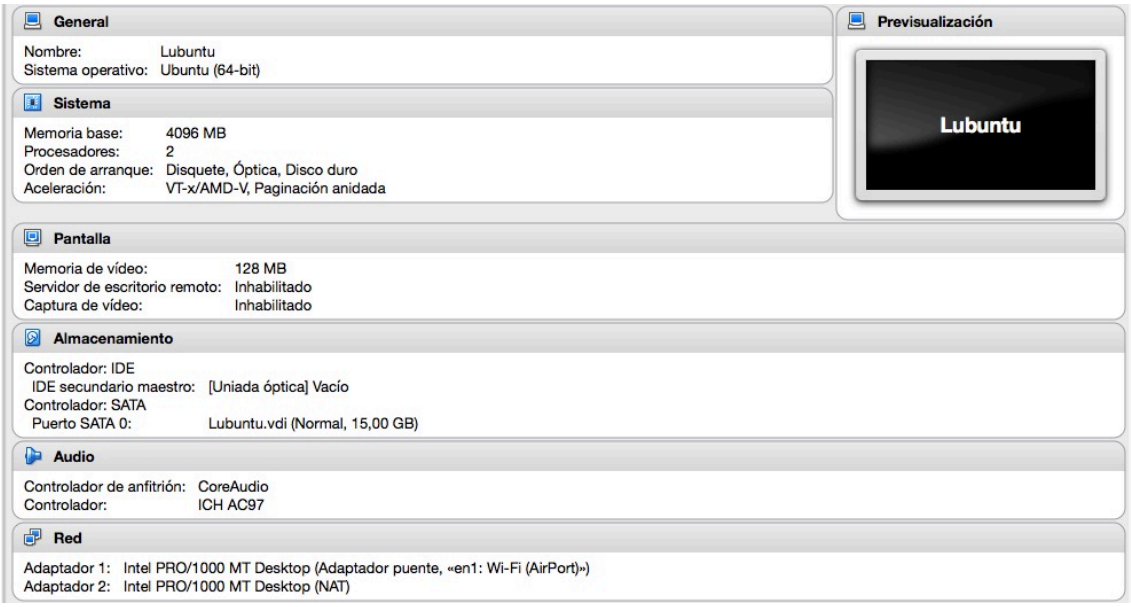


Figura 18: Configuración de la máquina virtual para el controlador Opendaylight

Fuente: Elaboración propia

En esta máquina hemos configurado dos interfaces de red, la primera en modo *bridge* para que se encuentre en la misma subred que la máquina virtual de Mininet, y la segunda en modo NAT para que esta máquina pueda salir a internet.

Una vez instalado el sistema operativo, descargamos de la web de Opendaylight [10] el controlador. En este TFG se utiliza la versión para desarrolladores de Lithium basada en Karaf, todo el proceso de instalación se encuentra en [18].

Lanzamos una ventana de terminal y ejecutamos el controlador, para ello nos desplazamos a la siguiente ruta: */integration/distribution/opendaylight-karaf/target/assembly/bin* y ejecutamos *./karaf*.

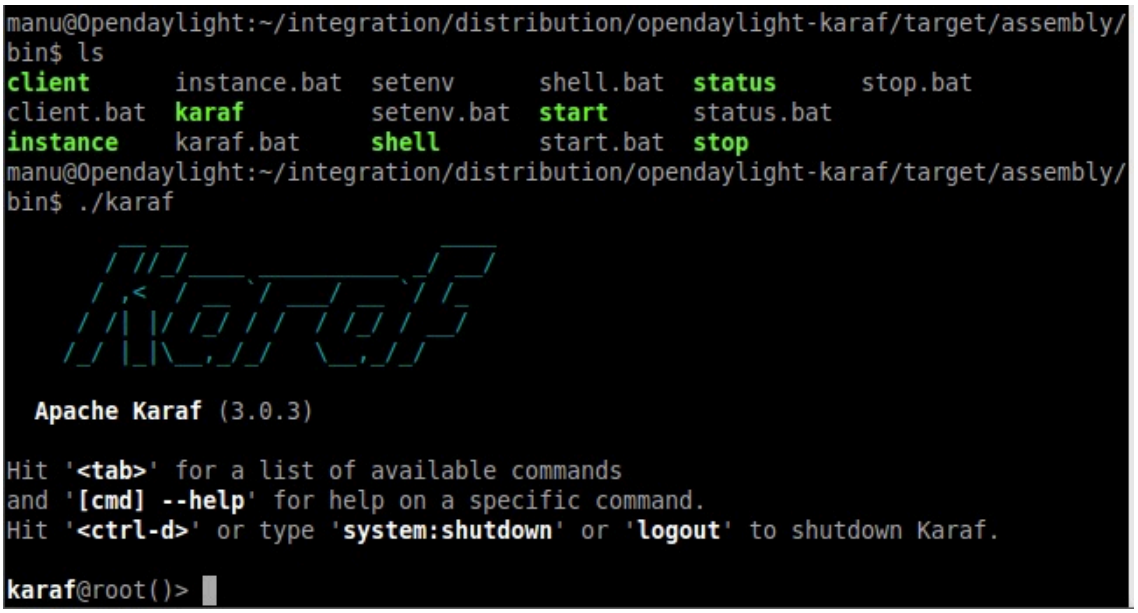


Figura 19: Ejecución del controlador Opendaylight

Fuente: Elaboración propia

Tras arrancar el controlador aparece el *prompt* de karaf con el usuario root, empezamos a instalar los *features* más importantes para el desarrollo del TFG. Los *features* instalados son:

- Odl-netconf: permite al controlador configurar y manipular los dispositivos conectados, por ejemplo, *switches*.
- Odl-dlux: permite acceder a la interfaz web del controlador
- Odl-openflowplugin: permite utilizar el puerto 6633 del protocolo OpenFlow
- Odl-mdsal: permite acceder mediante interfaz web a una lista de APIs de modelados de datos

Para instalar los *features* ejecutamos el siguiente comando:

feature:install odl-netconf-connector-all odl-openflowplugin-all odl-mdsal-all odl-dlux-all odl-adsal-northbound

Finalizada la ejecución del comando accedemos a la siguiente dirección desde el navegador de nuestra máquina virtual: 127.0.0.1:8181/index.html en ella se muestra la página de *login* del controlador.

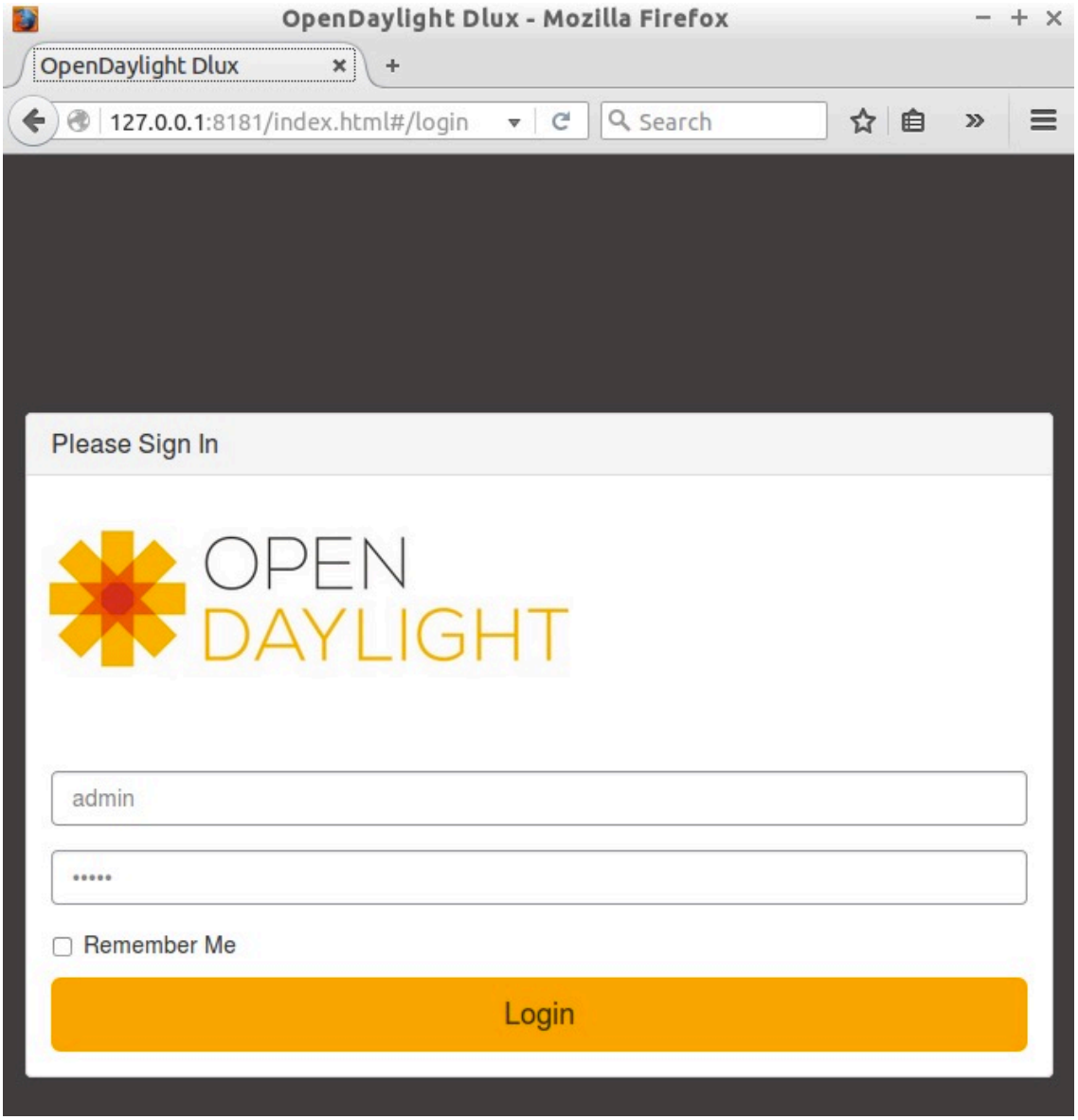


Figura 20: Login en la interfaz web del controlador Opendaylight

Fuente: Elaboración propia

Una vez introducidas las credenciales entramos a la siguiente página en la que ,si estuviera desplegada, veríamos la topología conectada a nuestro controlador, en el caso de la figura 20 no hay ninguna topología desplegada todavía y por tanto el panel está en blanco.

Configurado el controlador y visto el funcionamiento básico procedemos a la interconexión de la máquina con Mininet y el controlador.

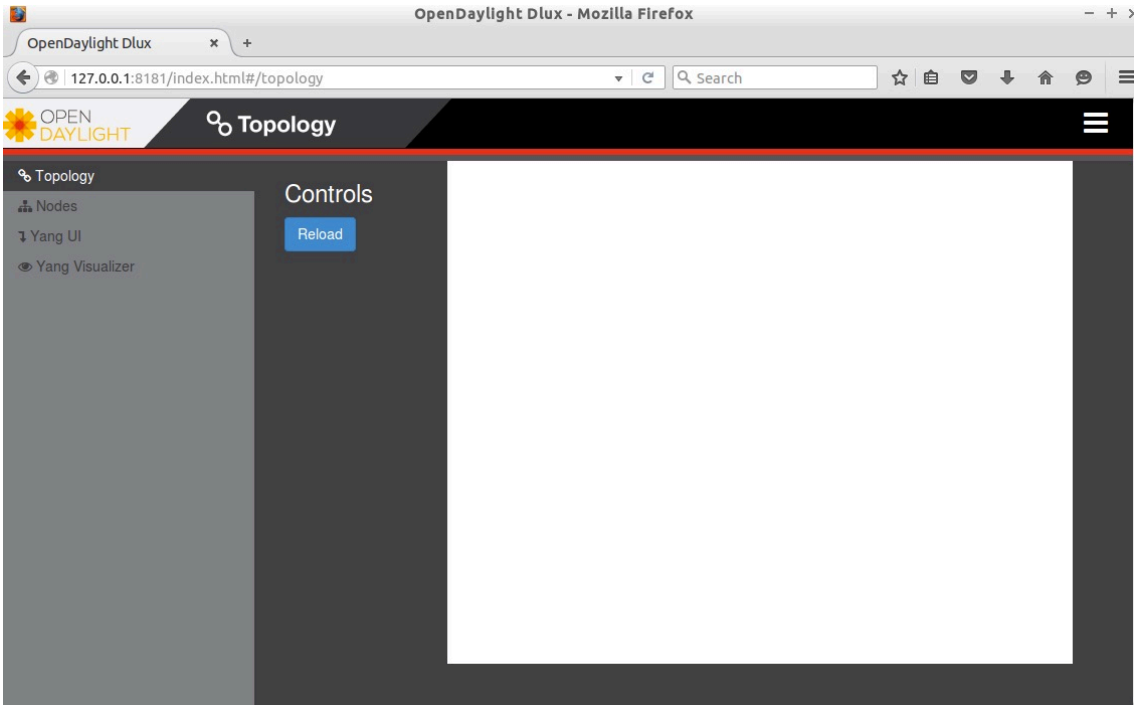


Figura 21:Interfaz web de Opendaylight dónde se verán las topologías desplegadas

Fuente: Elaboración propia

4.5.Interconexión del controlador y Mininet

Ejecutamos las dos máquinas virtuales, estando en funcionamiento, el primer paso es arrancar el controlador y ejecutar el comando *log:tail*, este comando devuelve el registro de eventos, en este caso muestra mensajes de inicialización de *features, bundles* y herramientas utilizadas por él.

```
karaf@root()> log:tail
2016-06-08 19:19:15,996 | INFO  | Event Dispatcher | RegionsPersistenceImpl
      | 49 - org.apache.karaf.region.persist - 3.0.3 | Loading region digraph pe
rsistence
2016-06-08 19:19:16,583 | INFO  | Event Dispatcher | SecurityUtils
      | 28 - org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using
the default JCE provider
2016-06-08 19:19:17,069 | INFO  | Event Dispatcher | core
      | 183 - org.apache.aries.jmx.core - 1.1.2 | Starting JMX OSGi agent
2016-06-08 19:19:17,120 | INFO  | Event Dispatcher | core
      | 183 - org.apache.aries.jmx.core - 1.1.2 | Registering MBean with ObjectN
ame [osgi.core:service=permissionadmin,version=1.2,framework=org.eclipse.osgi,uu
id=e09cbb19-9d2d-0016-1ec1-bdd139ae2375] for service with service.id [2]
2016-06-08 19:19:17,127 | INFO  | Event Dispatcher | core
      | 183 - org.apache.aries.jmx.core - 1.1.2 | Registering MBean with ObjectN
ame [osgi.compendium:service=cm,version=1.3,framework=org.eclipse.osgi,uuid=e09c
bb19-9d2d-0016-1ec1-bdd139ae2375] for service with service.id [37]
2016-06-08 19:19:17,180 | INFO  | Event Dispatcher | core
      | 183 - org.apache.aries.jmx.core - 1.1.2 | Registering org.osgi.jmx.frame
work.BundleStateMBean to MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@5143
f787 with name osgi.core:type=bundleState,version=1.7,framework=org.eclipse.osgi
,uuid=e09cbb19-9d2d-0016-1ec1-bdd139ae2375
```

Figura 22: Log del controlador Opendaylight

Fuente: Elaboración propia

Finalizados los mensajes del registro, en la máquina virtual de Mininet ejecutamos el siguiente comando: *sudo mn --controller=remote, ip=1.1.1.2 --topo single,2 --switch=ovsk, protocols=OpenFlow13*

El comando genera una topología con dos *host* h1 y h2, un *switch* s1 al que están ambos conectados, y conecta el *switch* al controlador remoto de nuestra máquina virtual. Del mismo modo fuerza que el protocolo utilizado de comunicación sea OpenFlow versión 1.3 y que el *switch* sea del tipo Open v*Switch*.

```
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=1.1.1.2 --topo single,2 -
witch=ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> _
```

Figura 23: Conexión de Mininet con el controlador remoto

Fuente: Elaboración propia

Del mismo modo que el registro del controlador muestra los mensajes de inicialización, también muestra las conexiones y desconexiones que se realizan. Creada la topología, ejecutamos el navegador de la máquina virtual de nuestro controlador y vamos a la dirección : *127.0.0.1:8181/index.html#/topology* y vemos como aparece el *switch* conectado al controlador.

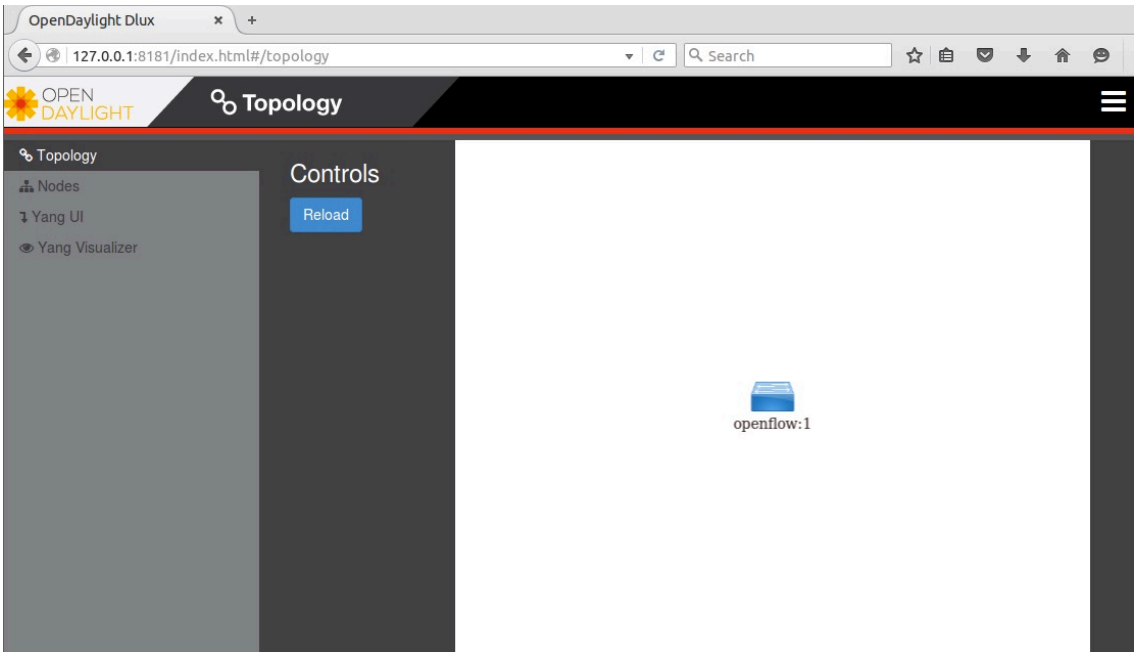


Figura 24: Visualización web del switch generado por Mininet conectado al controlador

Fuente: Elaboración propia

Para que el controlador muestre la topología completa debemos ejecutar en Mininet un comando *pingall* y probar la conectividad. Hecho esto, clickamos en "Reload" y vemos la topología completa desplegada, incluyendo las direcciones MAC de los *host*.

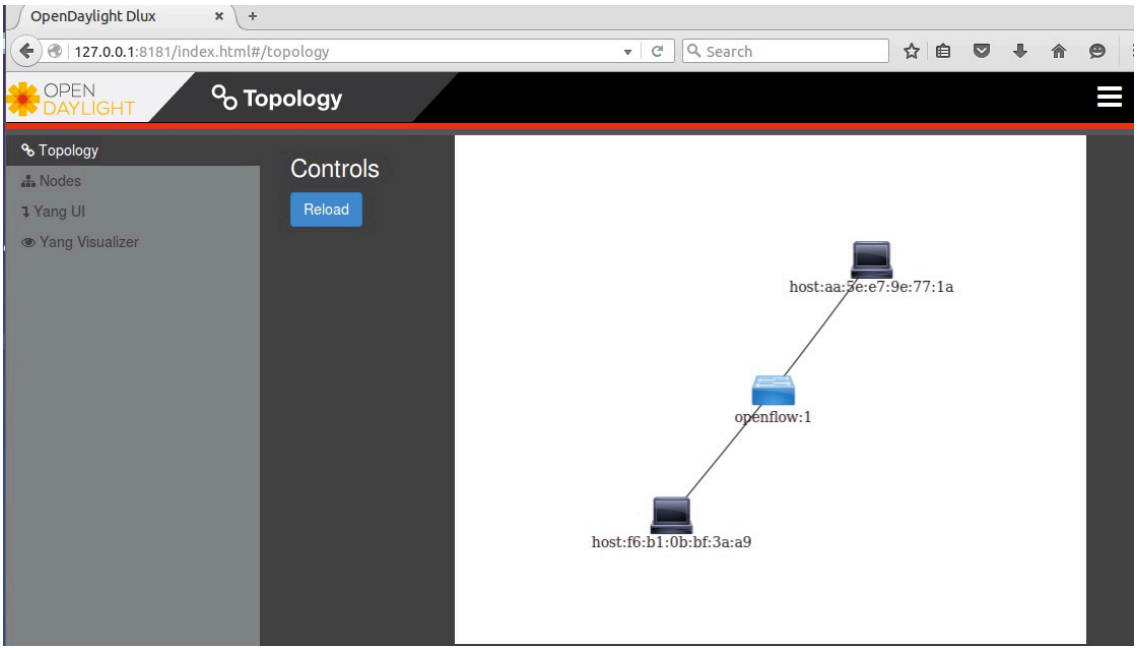


Figura 25: Topología desplegada por Mininet, vista desde la interfaz web del controlador

Fuente: Elaboración propia

Vista la interconexión entre ambas máquinas procedemos a presentar un esquema de la práctica.

4.6.Esquema de la primera parte de la práctica y pruebas

Una vez visto el funcionamiento básico del controlador, Mininet y la forma de interactuar entre ambos, procedemos a realizar la primera parte de la práctica. En esta parte generamos una topología de red personalizada y la conectamos al controlador.

Queremos que los alumnos vean como los flujos varían las condiciones de red; para ello desarrollamos un flujo que descarta paquetes en función de un filtrado MAC.

De este modo los alumnos verán como inicialmente existe conectividad entre todos los equipos, y una vez instalado el flujo ya no es así y por qué.

El esquema de la primera parte de la práctica se encuentra en la siguiente figura.

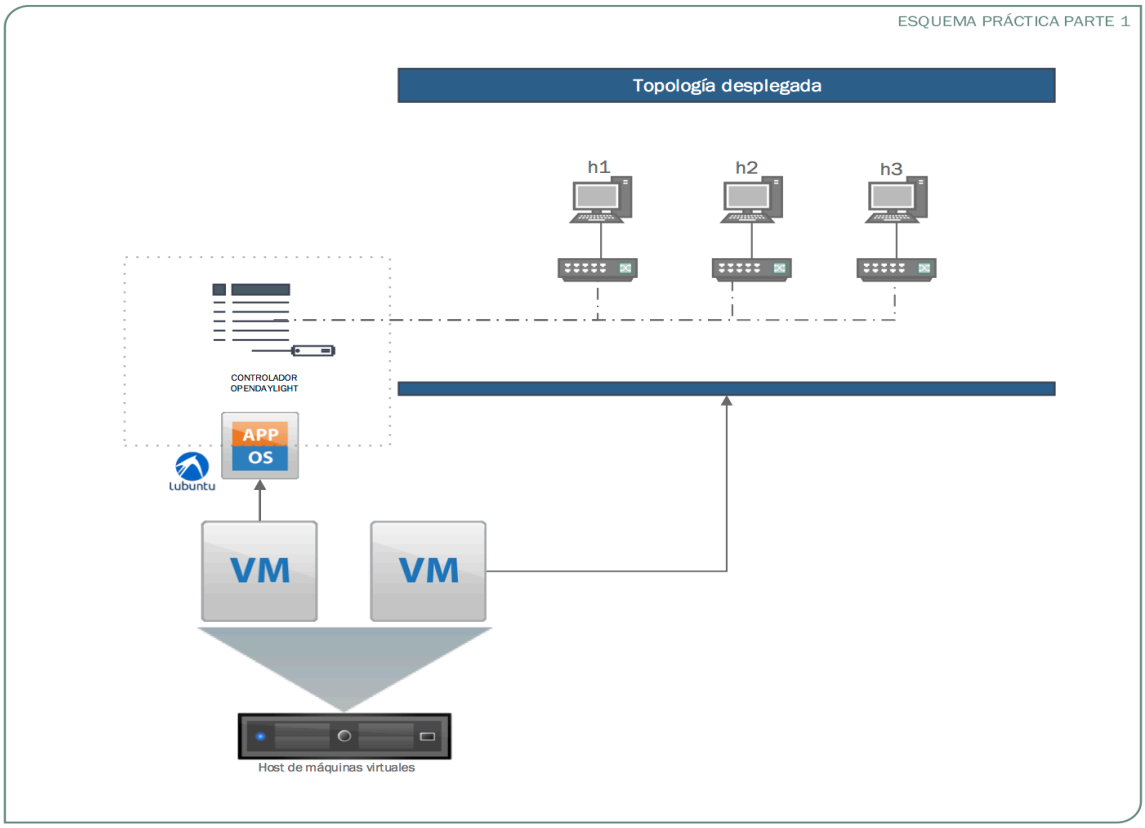


Figura 26: Diseño de parte 1 de la práctica

Fuente: Elaboración propia

Creada la topología comprobamos la conectividad entre *host* y procedemos a conectarla con el controlador.

Para este punto de la práctica realizamos un flujo, utilizando la REST API, que instalamos en uno de los *switches* y se encarga de descartar los paquetes que vayan de la MAC de h1 a la MAC de h2. El flujo completo se encuentra en el anexo A.3.

Conectados el controlador y la topología ejecutamos el comando que alojará el flujo programado en el *switch* indicado, en este caso el s1. El comando es el siguiente:

sudo curl --user admin:admin -i -X PUT -H "Content-Type: application/xml; charset=utf-8" -d @"/home/manu/Pruebas/Flujo1.xml" http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1

Utilizamos la opción *--user* para especificar el usuario y la contraseña del controlador, Opendaylight utiliza una autenticación básica HTTP, todas las peticiones a la interfaz REST *northbound* deben ser autenticadas.

Para añadir el flujo utilizamos una petición HTTP PUT, si existiese un flujo o recurso ya instalado con el mismo id, este sería modificado.

Especificamos el *content type* de HTTP como "application/xml" dado que vamos a enviar un fichero con formato XML.

Con la opción "-d" establecemos el *payload* de una petición HTTP, en este caso es un fichero XML que define el flujo que vamos a instalar en el *switch* virtualizado por Mininet.

El resto del comando incluye el almacén donde se va a instalar el flujo, en este caso en la tabla 0 del nodo OpenFLow con identificador 1.

A continuación se muestran las pruebas y el orden de ejecución de esta parte de la práctica:

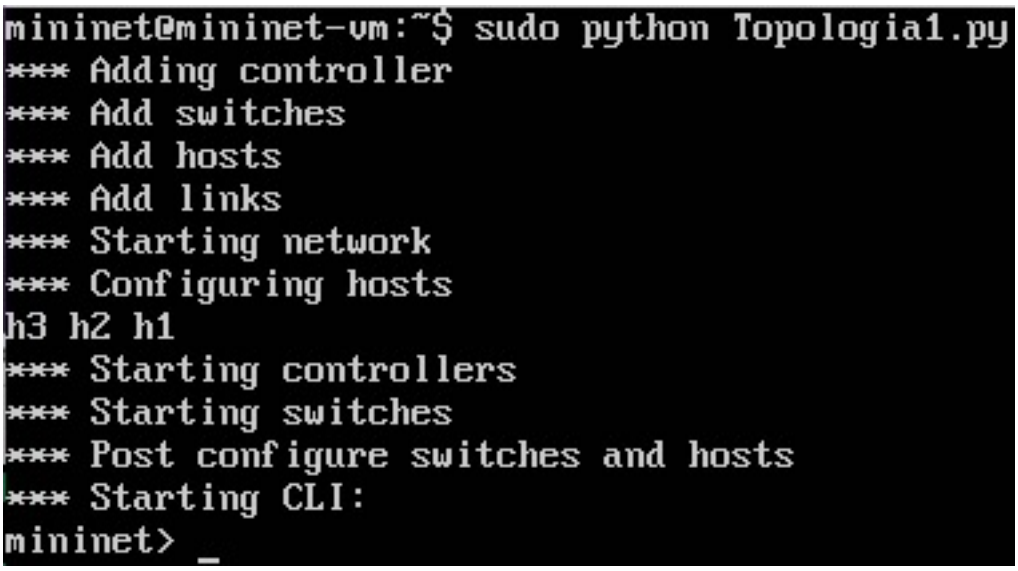


Figura 27: Ejecución de la topología personalizada

Fuente: Elaboración propia

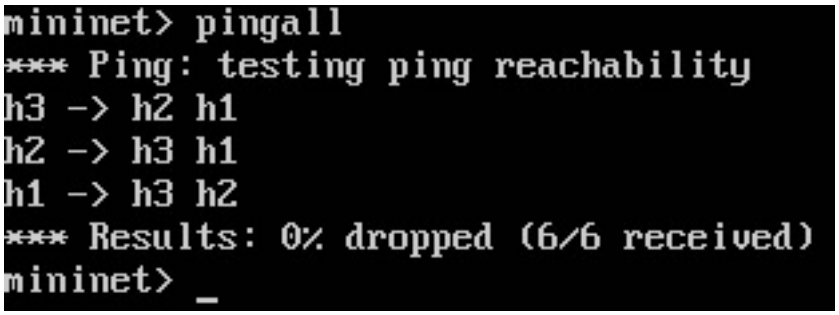


Figura 28: Prueba de conectividad entre host

Fuente: Elaboración propia

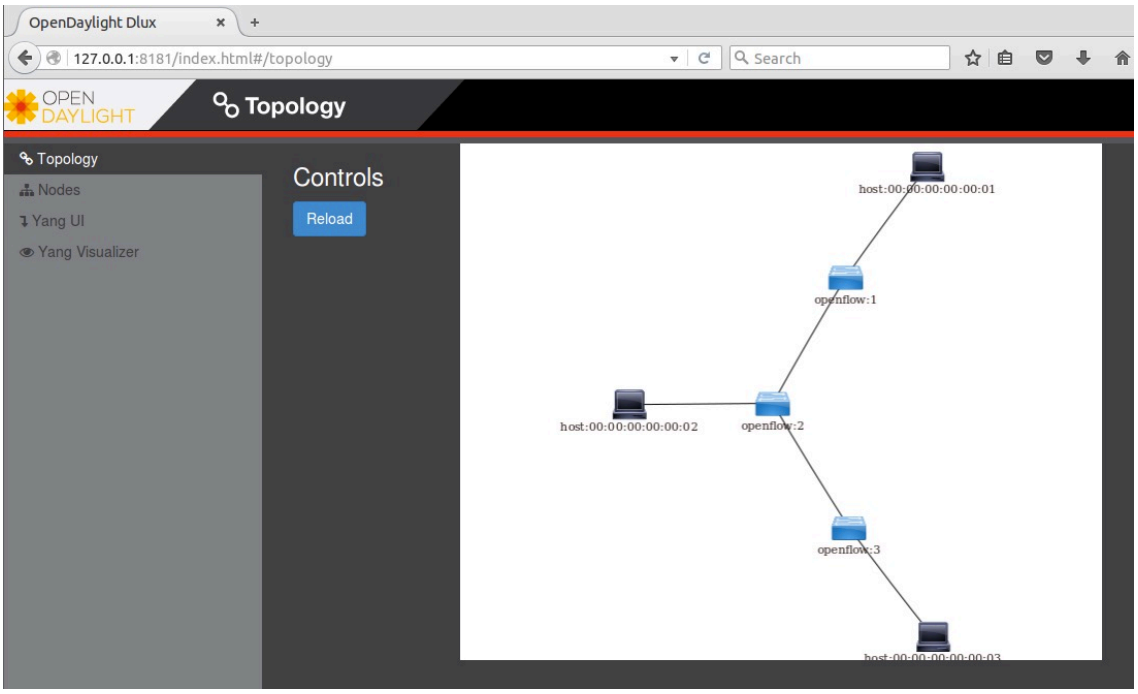


Figura 29: Prueba de la visión del controlador sobre la topología desplegada por Mininet

Fuente: Elaboración propia

```
manu@Opendaylight:~$ sudo curl --user admin:admin -i -X PUT -H "Content-Type: ap
plication/xml; charset=utf-8" -d @"/home/manu/Pruebas/Flujo1.xml" http://127.0.0
.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flo
w/1
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=1rhohn0astvd8nrhk3u0cjjj5;Path=/restconf
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: rememberMe=deleteMe; Path=/restconf; Max-Age=0; Expires=Wed, 08-Jun-
2016 18:02:38 GMT
Content-Length: 0
Server: Jetty(8.1.15.v20140411)

manu@Opendaylight:~$
```

Figura 30: Comando "curl" para enviar el flujo desarrollado al switch 1

Fuente: Elaboración propia

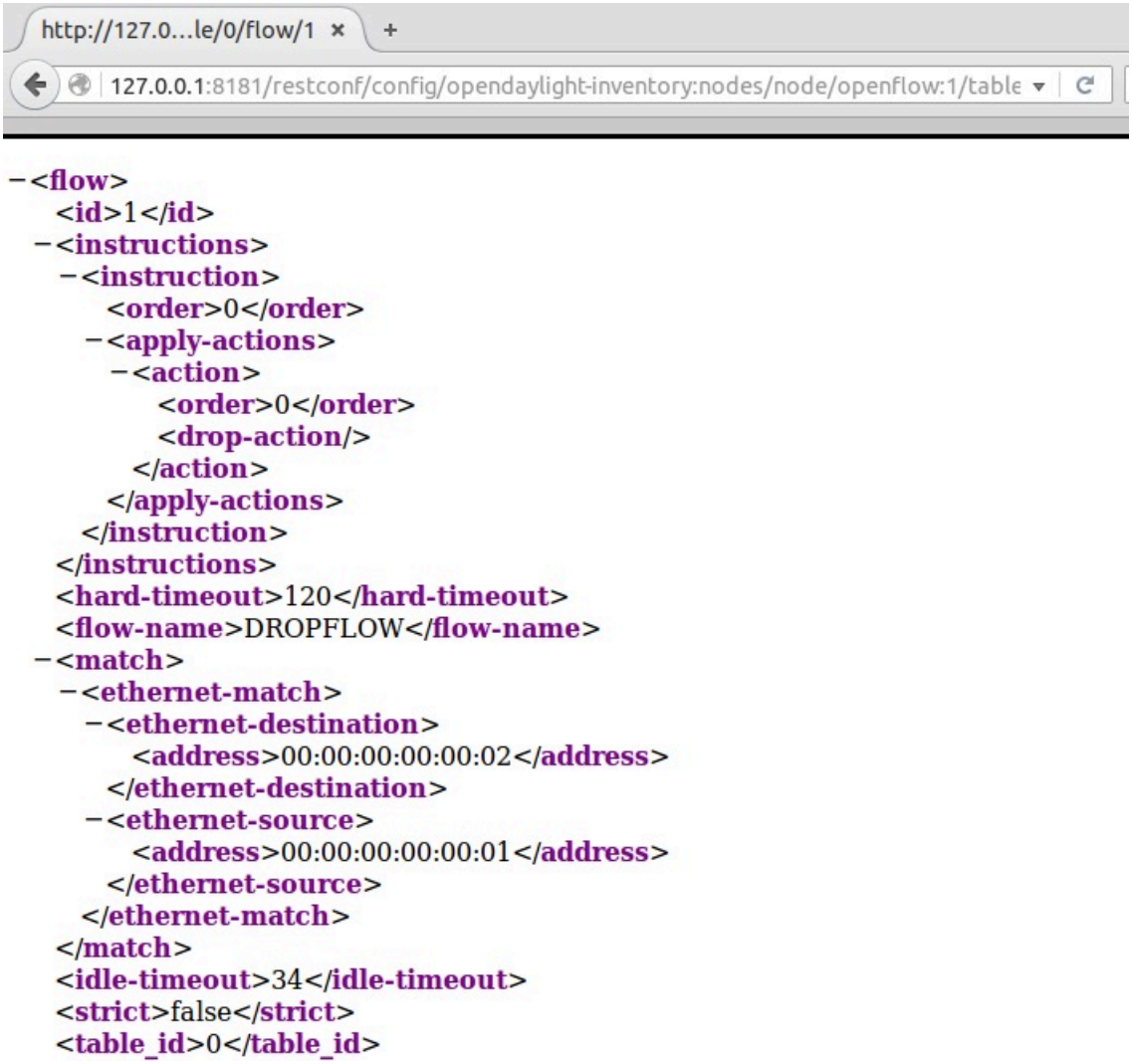


Figura 31: Comprobación de la instalación del flujo desde su almacén en el controlador

Fuente: Elaboración propia

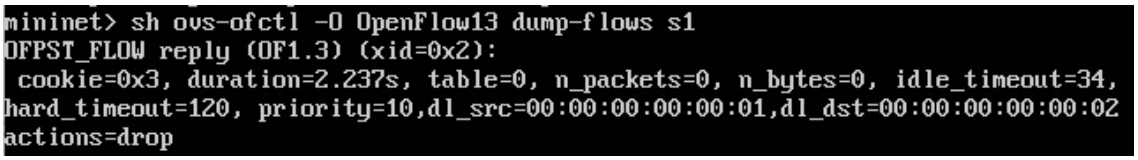


Figura 32: Prueba de instalación del flujo en el switch desde Mininet

Fuente: Elaboración propia

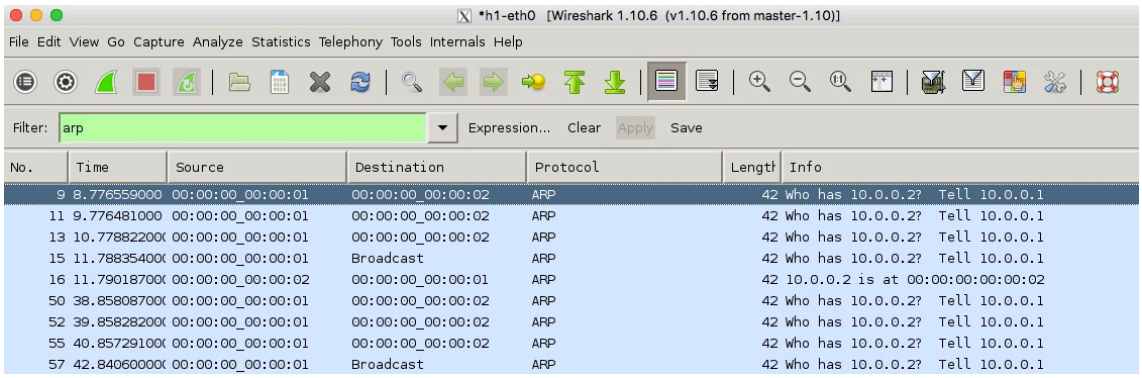


Figura 33: Captura de paquetes en h1

Fuente: Elaboración propia

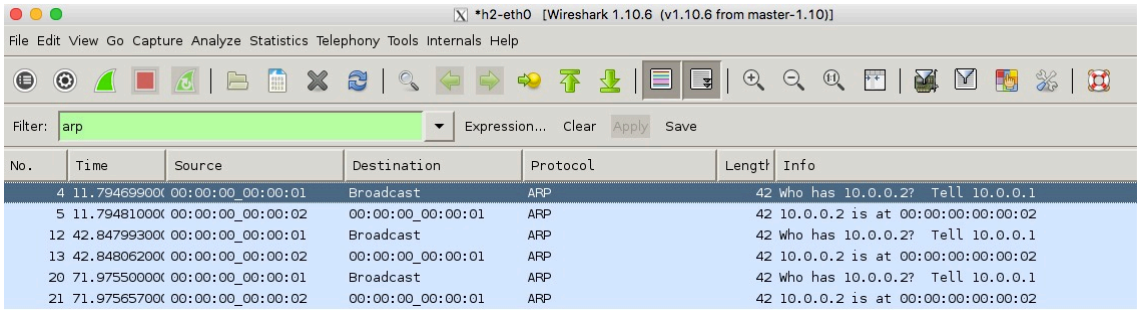


Figura 34: Captura de paquetes en h2

Fuente: Elaboración propia

Como vemos en las capturas de Wireshark los paquetes ARP enviados desde el *host* 1 con origen la MAC de h1 y destino la MAC de h2 no llegan al *host* 2. Esto se debe a que el flujo instalado está descartando los paquetes con ese filtro.

4.7.Esquema de la segunda parte de la práctica y pruebas

Hecha la primera parte de la práctica y visto su funcionamiento, procedemos a realizar la segunda parte.

En ella queremos qué los alumnos aprendan cómo instalar una aplicación en el controlador Opendaylight, entender su funcionamiento y utilizarla para instalar flujos desde la interfaz web del controlador. El funcionamiento de la aplicación consiste en filtrar los paquetes de un protocolo dado y reenviarlos por otro puerto.

A continuación se muestran el esquema de la segunda parte de la práctica y las pruebas realizadas sobre ella.

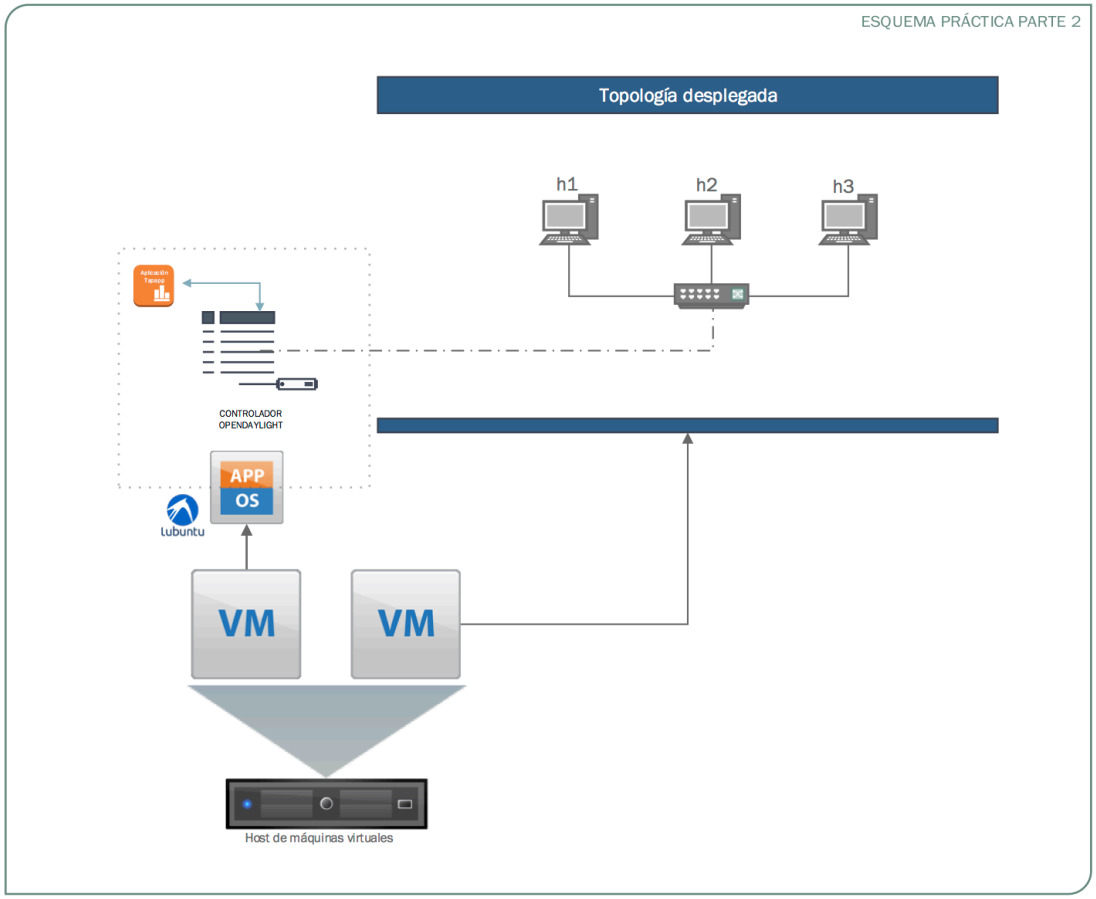


Figura 35: Diseño de la parte 2 de la práctica

Fuente: Elaboración propia

Los primeros pasos de configuración tanto de Mininet como del controlador son iguales que los de la primera parte de la práctica, realizados en el punto 4.6 de este Trabajo Fin de Grado.

Hecho esto, descargamos la aplicación realizada por el grupo SDN Hub [19] y alojamos las carpetas en */home/user/integration*. Para hacerla funcionar con nuestra configuración, editamos todos los ficheros *pom.xml,Java, Yang* y añadimos la ruta donde se encuentra la aplicación al fichero *features*.xml, para que al arrancar el controlador se inicialice la aplicación también. Esto es necesario para que el gestor de proyectos *Maven* conozca las dependencias entre los paquetes y pueda compilar el proyecto.

En el directorio */home/user/integration* ejecutamos el comando *mvn install -nsu;* la respuesta final es:

```
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ main ---
[INFO] Installing /home/manu/integration/pom.xml to /home/manu/.m2/repository/uc
3m/sdn/odl/tfg/main/0.6.0-SNAPSHOT/main-0.6.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] SDN common properties .............................. SUCCESS [ 16.002 s]
[INFO] SDN common utils ................................... SUCCESS [ 11.327 s]
[INFO] SDN Tap application Model .......................... SUCCESS [ 18.901 s]
[INFO] tapapp-parent ...................................... SUCCESS [  0.159 s]
[INFO] SDN Tap application Impl ........................... SUCCESS [ 37.758 s]
[INFO] SDN Tap application Config ......................... SUCCESS [  1.496 s]
[INFO] SDN features ....................................... SUCCESS [ 17.101 s]
[INFO] distribution-parent ................................ SUCCESS [  0.212 s]
[INFO] SDN distribution packaging ......................... SUCCESS [14:08 min]
[INFO] main ............................................... SUCCESS [  2.363 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16:11 min
[INFO] Finished at: 2016-06-10T20:09:40+02:00
[INFO] Final Memory: 108M/457M
[INFO] ------------------------------------------------------------------------
manu@Opendaylight:~/integration$
```

Figura 36: Utilización del comando "mvn" para compilar el proyecto

Fuente: Elaboración propia

Una vez finalizado la ejecución, arrancamos el controlador y vemos si se ha instalado la aplicación.

```
karaf@root()> feature:list | grep sdn
sdn-tapapp                                | 1.0.0-SNAPSHOT     | x         | f
eatures-1.0.0-SNAPSHOT                    | SDN :: OpenDaylight :: Tap applicati
on
```

Figura 37: Comprobación de la instalación de la aplicación en el controlador

Fuente: Elaboración propia

```
karaf@root()> bundle:list | grep sdn
178 | Active   |  80 | 1.0.0.SNAPSHOT                  | uc3m.sdn.odl.tfg.tap
app.tapapp-model
```

Figura 38: Comprobación del estado de la aplicación en el controlador

Fuente: Elaboración propia

Como vemos en las capturas anteriores la aplicación se encuentra instalada y activa.

Desarrollamos un pequeño fichero *JSON* que hará uso de la aplicación para filtrar y reenviar los paquetes del protocolo que se indique. El fichero se encuentra en el anexo

A.4. Del mismo modo que en la primera parte de la práctica, lo enviamos al *switch* con el comando *curl* por medio de una petición PUT HTTP.

A continuación se muestran capturas del funcionamiento y el orden de ejecución de la segunda parte de la práctica, una vez instalada la aplicación.

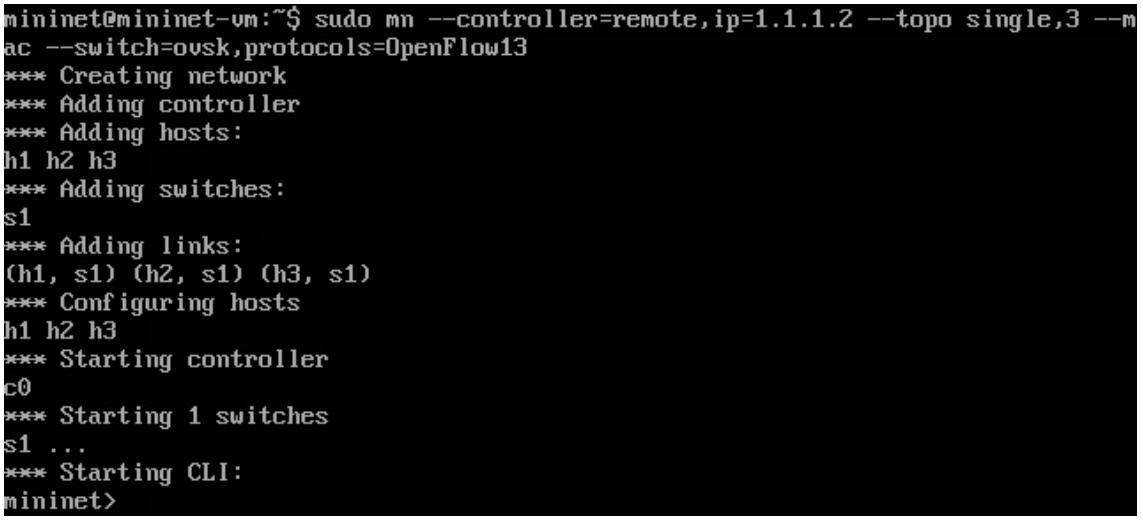


Figura 39: Creación de la topología de la parte dos de la práctica

Fuente: Elaboración propia

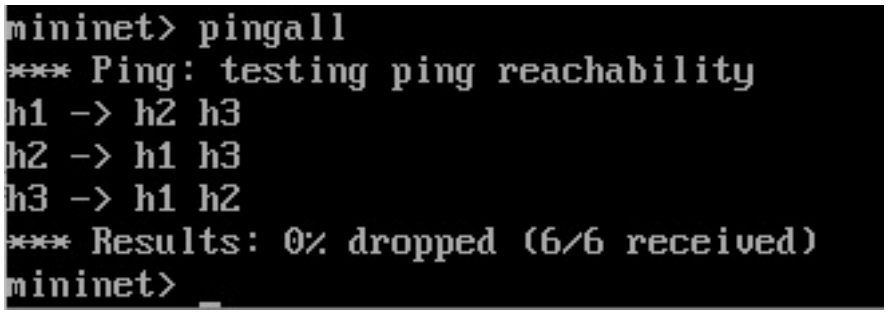


Figura 40: Prueba de conectividad

Fuente: Elaboración propia

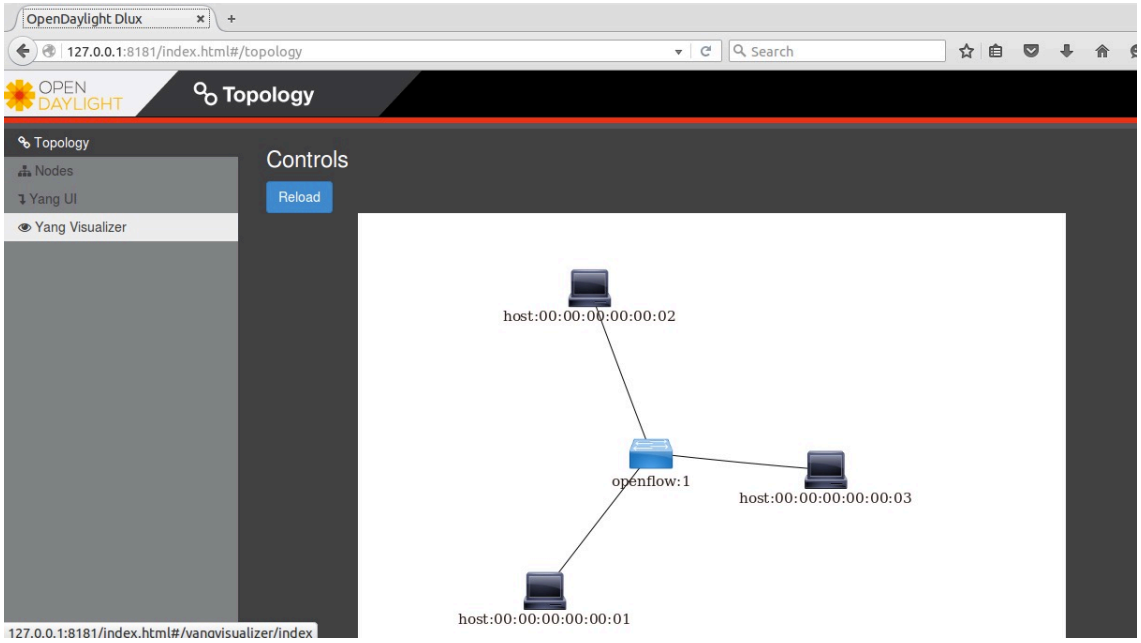


Figura 41: Vista de la topología desde el controlador

Fuente: Elaboración propia

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows -OOpenFlow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2a00000000000000, duration=392.037s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:01,dl_
dst=00:00:00:00:00:02 actions=output:2
 cookie=0x2a00000000000003, duration=392.005s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x2a00000000000001, duration=392.037s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:02,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x2a00000000000005, duration=391.934s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:02 actions=output:2
 cookie=0x2a00000000000002, duration=392.005s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:01,dl_
dst=00:00:00:00:00:03 actions=output:3
 cookie=0x2a00000000000004, duration=391.934s, table=0, n_packets=2, n_bytes=140
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:02,dl_
dst=00:00:00:00:00:03 actions=output:3
 cookie=0x2b00000000000000, duration=392.673s, table=0, n_packets=4, n_bytes=280
, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
 cookie=0x2b00000000000002, duration=392.667s, table=0, n_packets=7, n_bytes=406
, priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=392.669s, table=0, n_packets=4, n_bytes=280
, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
```

Figura 42: Flujos instalados automáticamente por el controlador

Fuente: Elaboración propia

En este punto tenemos conectividad entre todos los host y los flujos de reenvío instalados por el controlador al ejecutar por primera vez un *pingall*. A partir de aquí se muestra el funcionamiento desde la instalación del flujo.

```
manu@Opendaylight:~/integration/distribution/opendaylight-karaf/target/assembly/
bin$ sudo curl --user admin:admin -i -X PUT -H "Content-Type: application/json;
charset=utf-8" -d @"/home/manu/integration/tapapp/postman-resources/tapapp.json"
 http://127.0.0.1:8181/restconf/config/tap:tap-spec
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=6hmp8kx56439avdcp7o1ekub;Path=/restconf
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: rememberMe=deleteMe; Path=/restconf; Max-Age=0; Expires=Mon, 06-Jun-
2016 19:31:47 GMT
Content-Length: 0
Server: Jetty(8.1.15.v20140411)
```

Figura 43: Instalación del flujo mediante comando curl

Fuente: Elaboración propia

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2a00000000000000, duration=186.874s, table=0, n_packets=4, n_bytes=336
, idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:01,dl_
dst=00:00:00:00:00:02 actions=output:2
 cookie=0x2a00000000000003, duration=186.859s, table=0, n_packets=1, n_bytes=42,
 idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:03,dl_d
st=00:00:00:00:00:01 actions=output:1
 cookie=0x2a00000000000001, duration=186.873s, table=0, n_packets=1, n_bytes=42,
 idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:02,dl_d
st=00:00:00:00:00:01 actions=output:1
 cookie=0x2a00000000000005, duration=186.855s, table=0, n_packets=1, n_bytes=42,
 idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:03,dl_d
st=00:00:00:00:00:02 actions=output:2
 cookie=0x2a00000000000002, duration=186.861s, table=0, n_packets=1, n_bytes=42,
 idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:03 actions=output:3
 cookie=0x2a00000000000004, duration=186.855s, table=0, n_packets=1, n_bytes=42,
 idle_timeout=1800, hard_timeout=3600, priority=10,dl_src=00:00:00:00:00:02,dl_d
st=00:00:00:00:00:03 actions=output:3
 cookie=0x2b00000000000000, duration=194.46s, table=0, n_packets=10, n_bytes=812
, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
 cookie=0x0, duration=15.753s, table=0, n_packets=0, n_bytes=0, in_port=1 action
s=output:2
 cookie=0x2b00000000000002, duration=194.46s, table=0, n_packets=7, n_bytes=518,
 priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=194.46s, table=0, n_packets=10, n_bytes=812
, priority=2,in_port=2 actions=output:3,output:1,CONTROLLER:65535
 cookie=0x2b00000000000001, duration=198.236s, table=0, n_packets=0, n_bytes=0,
priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000001, duration=198.236s, table=0, n_packets=0, n_bytes=0,
priority=0 actions=drop
```

Figura 44: Comprobación de la instalación del flujo cookie oxo

Fuente: Elaboración propia

Figura 45: Comprobación de la instalación del flujo en el almacén del controlador

Fuente: Elaboración propia

Figura 46: Ping de h1 a h3 para probar el funcionamiento del flujo

Fuente: elaboración propia

Figura 47: Detalle de la instalación del flujo

Fuente: Elaboración propia

En la última captura, al haber instalado el flujo, ya no existe conectividad entre h1 y h3. Esto se debe a que todos los paquetes que llegan al puerto 1 del *switch* son encaminados a la salida 2, que en este caso es el *sink node connector*. Todos los paquetes que tengan como origen la entrada 1 son reenviados al puerto 2.

En la figura 47 observamos cómo el campo *n_packets* se ha incrementado a raiz del *ping* entre h1 y h3. Esto es debido a que el flujo está filtrando los paquetes del protocolo ARP.

Capturando con Wireshark el tráfico generado en h2 vemos que todos los *reply* de h1 llegan a h2 y no a h3, como debería ser. Esto es porque los paquetes procedentes de h1 tienen como origen el puerto 1 del switch y como hemos descrito arriba, todos los paquetes con origen el puerto 1 son reenviados al puerto 2.

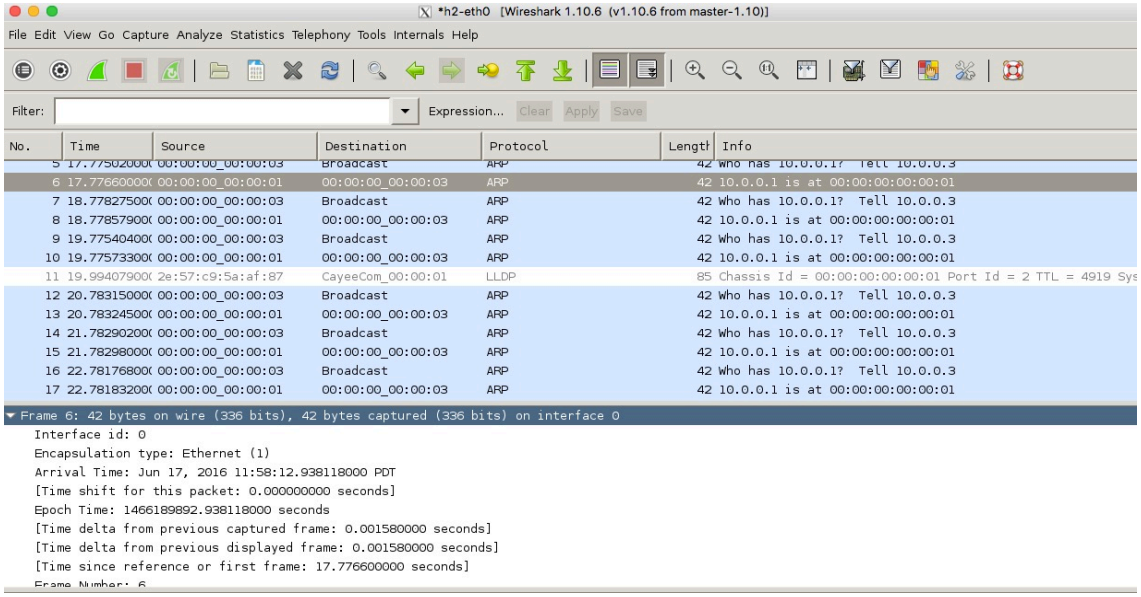


Figura 48: Traza de Wireshark: h3 ping h1. Capturada en h2

Fuente: Elaboración propia

Los primeros mensajes de la traza, en la figura 46, son las respuestas ARP de h1 a h3. Al no obtener respuesta de h3, el *host* 1, dado que al *host* 3 no le están llegando los paquetes, empieza emitir a *Broadcast*, es h2 el que se queda con todo el tráfico procedente de h1.

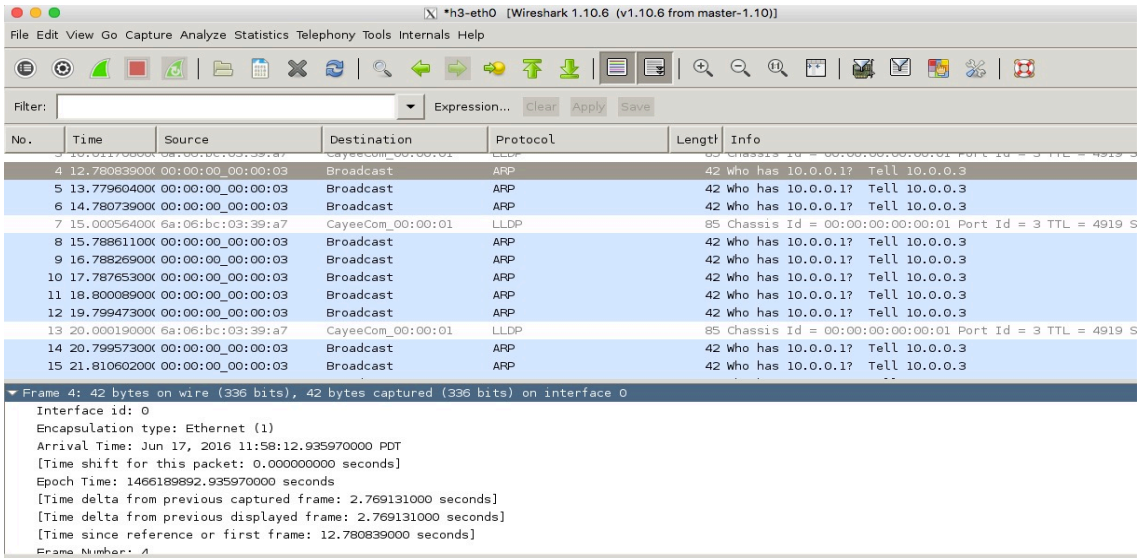


Figura 49: Captura de paquetes en h3

Fuente: Elaboración propia

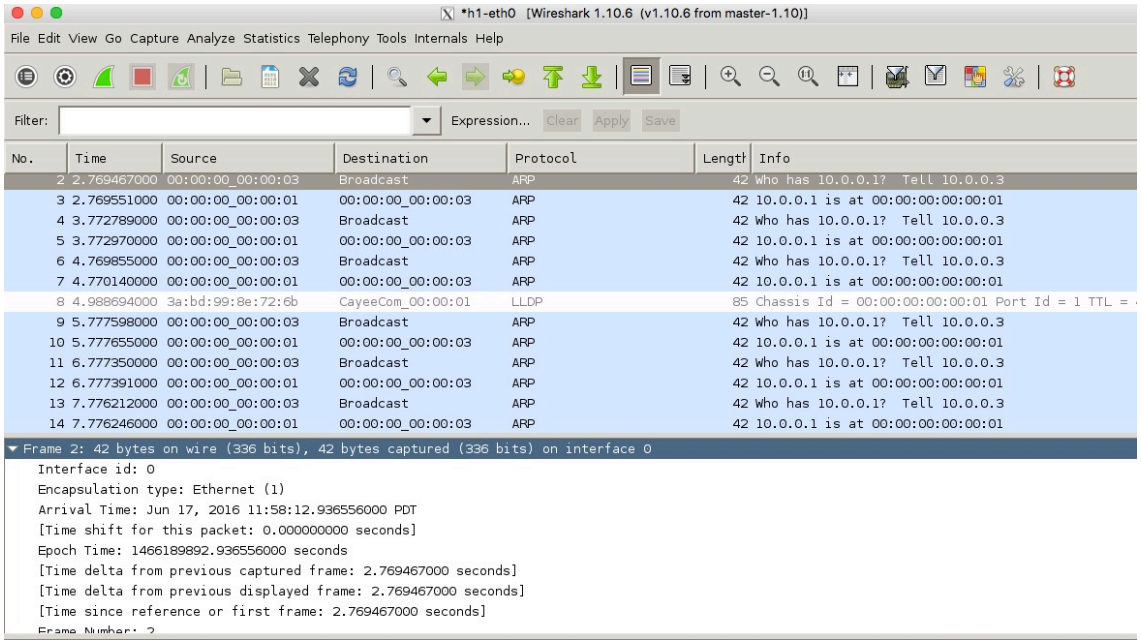


Figura 50: Captura de paquetes en h1

Fuente: Elaboración propia

En la figura 48 vemos cómo a h1 si que le llegan los mensajes con origen h3, y cómo h1 contesta.

Gracias a la aplicación podemos instalar el flujo desde la interfaz web de Opendaylight, para ello en el navegador web introducimos la siguiente dirección:

127.0.0.1:8181/index.html

A continuación seleccionamos la opción Yang UI en la barra lateral y navegamos por su menú (figura 44). Elegimos *tap rev* y en el menú desplegable elegimos *configuration* una vez aparezcan las opciones de configuración rellenamos los campos y clickamos en *Send*. Hecho esto el flujo funciona de la misma manera que si lo instalamos con el comando *curl*.

Desde la interfaz web es más sencillo instalar, eliminar y modificar los flujos.

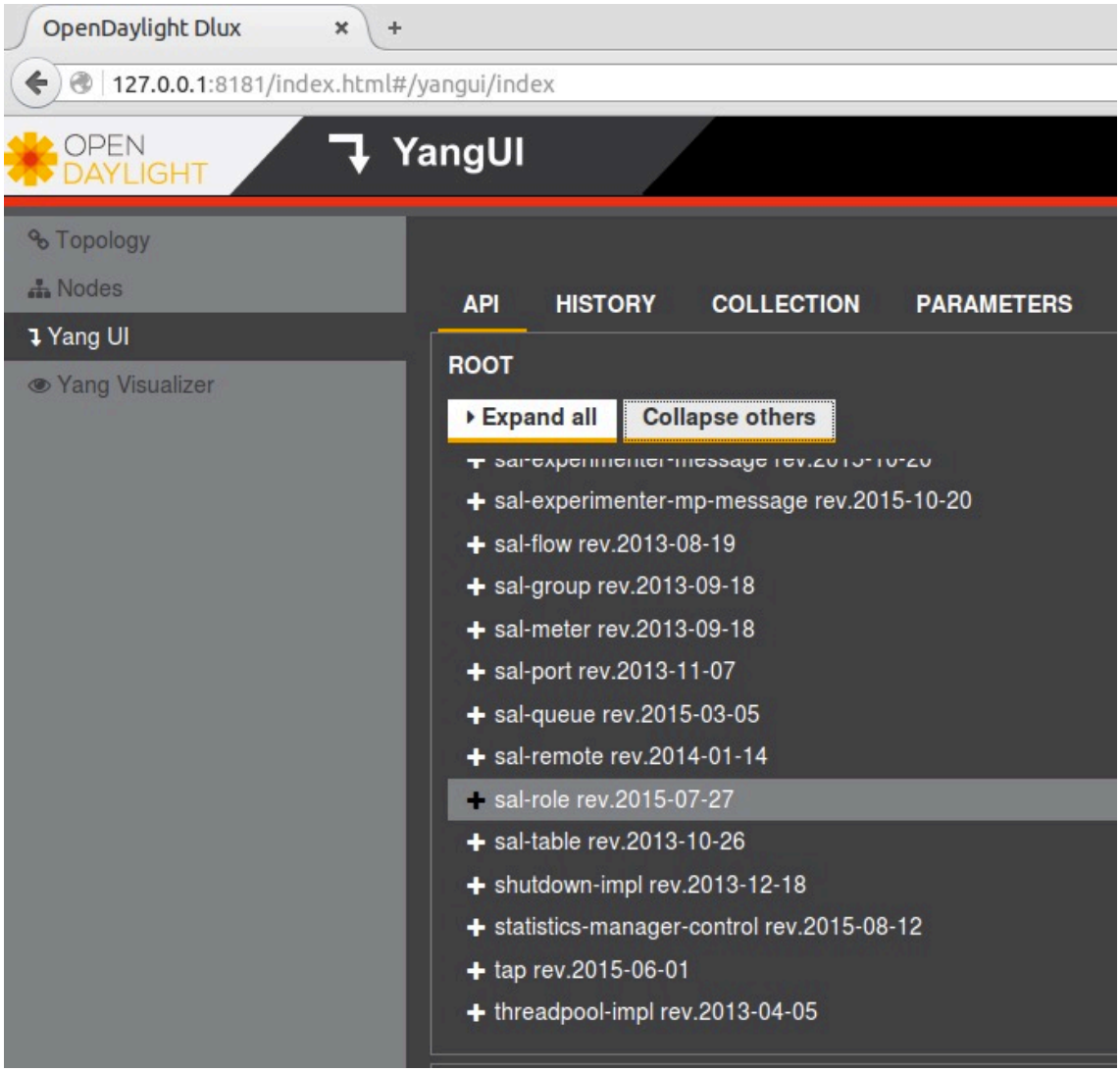


Figura 51: Menú del Yang UI

Fuente: Elaboración propia

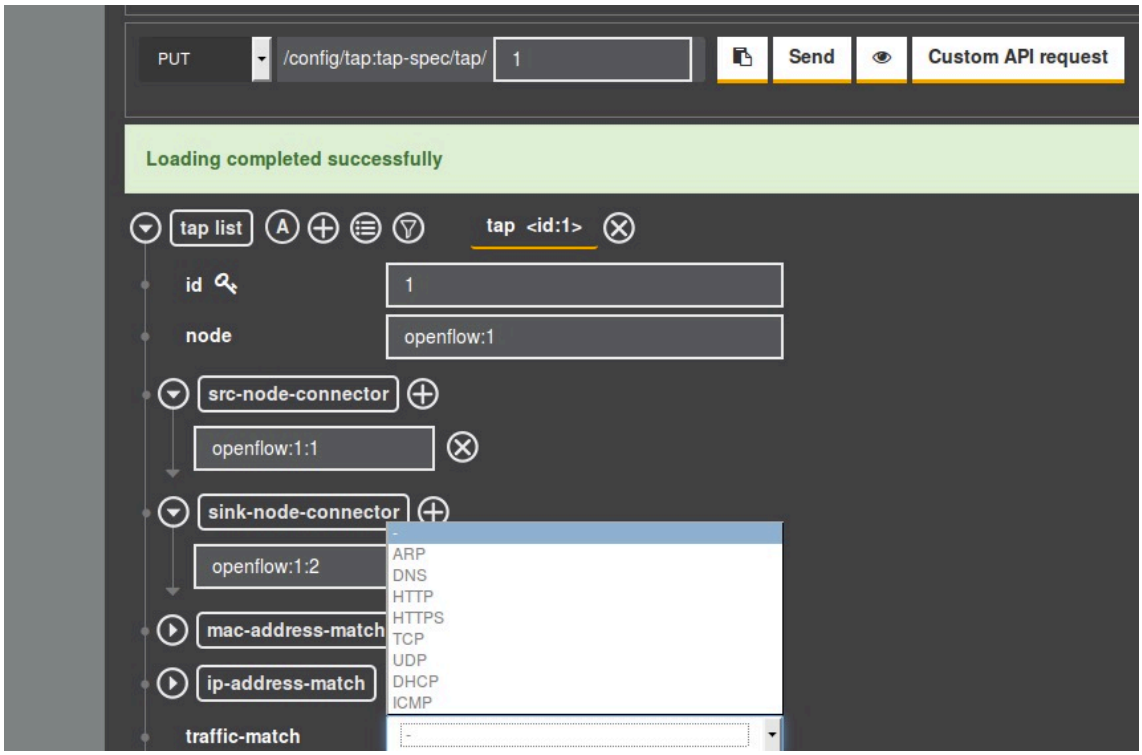


Figura 52: Plantilla generada por la aplicación para crear los flujos

Fuente: Elaboración propia

Figura 53: Éxito al enviar el flujo a S1

Fuente: Elaboración propia

5. GUION DE PRÁCTICAS

5.1. Introducción y objetivos

El objetivo de este laboratorio es conocer los elementos principales explicados en el curso de SDN. El tiempo estimado de este laboratorio es de 8 horas por lo que no todo podrá ser revisado, por ello nos centraremos en las siguientes tareas:

- Mininet
  - → Conocer comandos básicos de Mininet
  - → Crear topologías personalizadas
- Opendaylight
  - → Familiarizarse con el controlador Opendaylight y sus funcionalidades
- Creación de flujos OpenFlow
  - → Desarrollar flujos OpenFlow para instalar en los switches
  - → Ver el funcionamiento de los flujos con ejemplos
- Instalación de una aplicación en el controlador
  - → Instalar una aplicación de filtrado de paquetes
  - → Ejecutar flujos basados en la aplicación

5.1.1. Directrices

Lea atentamente este documento hasta el final antes de comenzar el laboratorio.
Si tiene alguna duda, acuda al profesor.
Cada vez que finalice un *hito* avise al profesor para que se lo evalúe.
Duración estimada del laboratorio: 8 horas.

5.1.2. Entorno de trabajo

El entorno de trabajo está compuesto por un equipo del laboratorio, y dos máquinas virtuales desplegadas en él, con las que el alumno deberá trabajar. Es conveniente conectarse vía *ssh* a la máquina virtual de Mininet.

Una de las máquinas contiene un sistema operativo Linux con el controlador Opendaylight instalado. La otra, como ya se ha mencionado, contiene la herramienta de virtualización de red, Mininet.

Compruebe que ambas máquinas virtuales tienen configuradas dos interfaces de red, la primera como *Bridge* y la segunda como *Host-Only*, si no es así, configúrelas.

5.2. Parte I: Flujo XML (tiempo estimado de 4 horas)

1.1. Introducción

Antes de comenzar, es necesario establecer una conexión ssh con la máquina virtual Mininet:

```
portátil$ ssh –Y mininet@IP_De_La_Máquina_Virtual

Las credenciales son: login: mininet password: mininet
```

Todas las máquinas utilizadas en este laboratorio tienen instalados los paquetes necesarios para ejecutar todo lo necesario.

1.2. Mininet

Vamos a comenzar conociendo comandos básicos de Mininet, para ello desde el terminal con acceso remoto a Mininet, vamos a ejecutar los siguientes comandos y ver su funcionamiento:

- `sudo mn`
- `nodes`
- `net`
- `dump`
- `h1 ifconfig -a`
- `s1 ifconfig -a`
- `pingall`
- `xterm h1`

Hito 1. Llegados a este punto debería tener desplegada una red con 2 *hosts*, 1 *switch*, un controlador local y todo ello con conectividad. Responda a las siguientes preguntas: ¿qué realiza cada comando? ¿qué diferencias encuentra entre las interfaces que aparecen al realizar el comando *h1 ifconfig -a* y las que aparecen al realizar *s1 ifconfig -a*? ¿Por qué esas diferencias?

1.3. Opendaylight

Para ejecutar el controlador Opendaylight diríjase a la máquina virtual con sistema operativo Linux, lance un terminal e introduzca este comando:

```
cd/home/USER_X/integration/distributions/opendaylight/target/assembly/bin
```

Hecho esto debería poder ejecutar: *./karaf* Una vez aparezca el *prompt* de karaf instale los *features* necesarios para la práctica. Para ello ejecute:

```
karaf@root>feature:install odl-netconf-all odl-l2switch-connector-all odl-mdsal-all odl-dlux-all
```

Una vez instalados ejecute *feature:list* y compruebe si se han instalado correctamente las características anteriores.

Lance el navegador web instalado en la máquina virtual y diríjase a la siguiente URL:

```
127.0.0.1:8181/index.html
```

Cuando aparezca el *login* de Opendaylight introduzca las siguientes credenciales:

```
Usuario: admin
Contraseña: admin
```

1.4. Interconexión de máquinas virtuales

Conocidos algunos comandos de Mininet y configurado el controlador, vamos a proceder a desplegar una topología en Mininet que utilice el controlador Opendaylight.

En Mininet podemos crear topologías con controladores remotos, para ello debemos indicarlo al desplegar la red.

Hito 2. Despliegue una topología básica que utilice el controlador Opendaylight ejecutado en la máquina virtual. Si tiene alguna duda siempre puede acudir al manual del comando: `sudo mn --help`

Con la conexión realizada diríjase al controlador Opendaylight y en la interfaz web haga click en “Reload”

Hito 3. ¿Qué ve?. Ejecute el comando *pingall* en Mininet, vuelva al controlador y seleccione de nuevo “Reload”. ¿Qué ve ahora?. ¿Por qué ha ocurrido este cambio?.

Lance un nuevo *ssh* a Mininet y ejecute el siguiente comando: `sudo ovs-ofctl dump-flows -OOpenFlow13 s1`. Este comando muestra los flujos del protocolo OpenFlow instalados en el *switch* 1.

Detenga la topología creada anteriormente y elimínela ejecutando: `sudo mn -c`. Hecho esto cree una nueva topología nueva y en la nueva ventana con ssh ejecute el comando para ver los flujos instalados

Hito 4. ¿Qué ve al ejecutar el comando?. Haga un *pingall,* ¿qué ocurre si vuelve a mostrar los flujos? ¿por qué ocurre esto?.

Hito 5. Elimine de nuevo la topología y cree una nueva. Ejecute el siguiente comando 2 veces seguidas y explique por qué el tiempo de llegada de los paquetes del segundo *ping* es más corto que los del primero: h1 `ping -c 1 h2`.

1.5. Creación de una topología personalizada

A continuación queremos desplegar una topología personalizada. Para ello ejecute lo siguiente desde el terminal de Mininet:

```
mininet@mininet-vm:~$ sudo ~/mininet/examples/miniedit.py
```

Hito 6. En el editor de topologías diseñe lo siguiente:

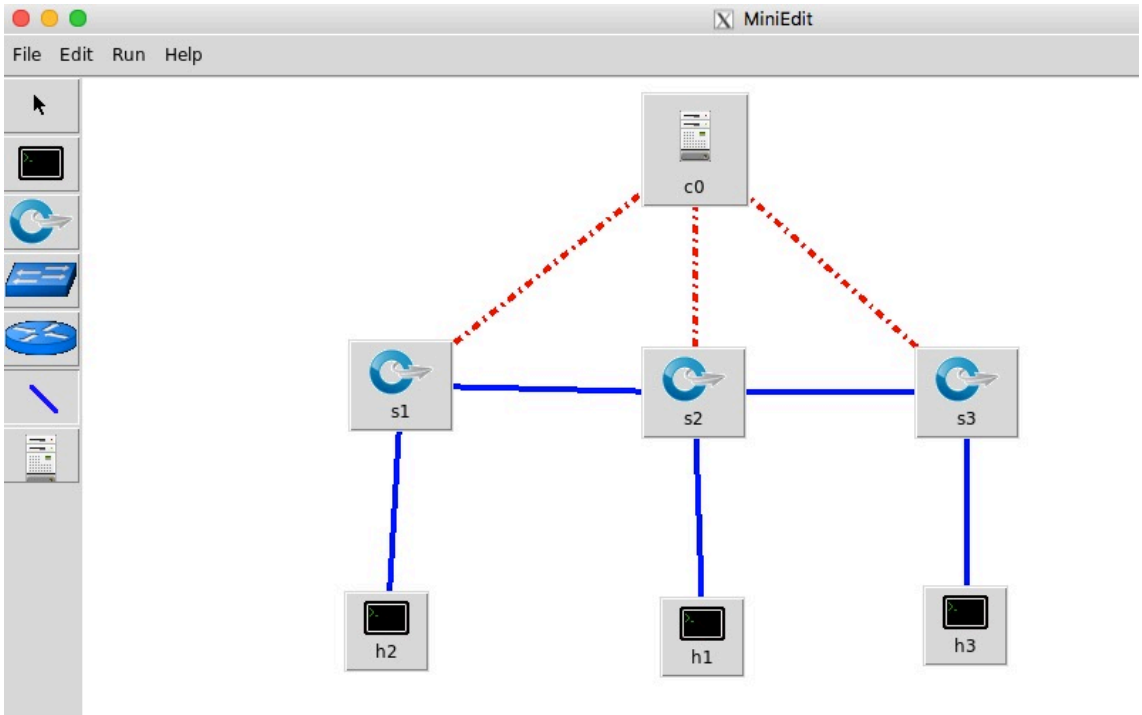


Figura 54: Topología manual parte 1

Fuente: Elaboración propia

Una vez creada, para guardarla vaya a "File"→"Export Level 2 Script" . Esto generará un fichero python que más adelante podremos editar.

Hito 7. Abra el fichero *xxx.py* generado y edítelo con la siguiente configuración:

- ListenPort: 6633
- ipBase: 10.0.0.0/8
- controller=RemoteController
- ip='Ip_de_la_máquina_virtual_del_Controlador'
- Configure las IPs y las MACs de los host de la siguiente forma:
  - → H1: 10.0.0.1 00:00:00:00:00:01
  - → H2: 10.0.0.2 00:00:00:00:00:02
  - → H3: 10.0.0.3 00:00:00:00:00:03
- Configure las MACs de los switches de la siguiente forma:
  - → S1: 00:00:00:00:00:04
  - → S2: 00:00:00:00:00:05
  - → S3: 00:00:00:00:00:06

Guarde la configuración y ejecute: `sudo python nombre_de_topología.py`

Hito 8. Compruebe la conectividad de la topología creada.

1.6. Desarrollo e instalación de un flujo personalizado

Como hemos visto antes, el controlador envía flujos a los switches para configurar directrices de funcionamiento. En este apartado queremos generar un flujo que descarte los paquetes que tenga MAC origen la del *host* 1 y MAC destino la del *host* 2. El flujo deberá instalarse en el switch 1.

Hito 9. Rellene los campos con XXX de la plantilla para cumplimentar el flujo especificado.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
     <instructions>
          <instruction>
               <order>0</order>
               <apply-actions>
                    <action>
                       <order>0</order>
                    <drop-action/>
                    </action>
               </apply-actions>
          </instruction>
     </instructions>
          <table_id>XXX</table_id>
```

```
            <id>XXX</id>
            <cookie_mask>255</cookie_mask>
            <installHw>false</installHw>
            <match>
              <ethernet-match>
              <ethernet-destination>>
                 <address>XXX</address>
              </ethernet-destination>
              <ethernet-source>
                 <address>XXX</address>
              </ethernet-source>
              </ethernet-match>
            </match>
            <hard-timeout>XXX</hard-timeout>
            <cookie>XXX</cookie>
            <idle-timeout>34</idle-timeout>
            <flow-name>Parte1</flow-name>
            <priority>XXX</priority>
            <barrier>false</barrier>
</flow>
```

Guarde el fichero generado con extensión .xml.

Cree una nueva ventana de terminal en la máquina virtual del controlador y proceda con la instalación del flujo en el *switch* 1 de la topología creada anteriormente. Para instalar el flujo deberá ejecutar el siguiente comando.

*sudo curl --user admin:admin -i -X PUT -H "Content-Type: application/xml; charset=utf-8" -d @"/home/**USER_X**/**CARPETA_DE_SU_FICHERO**/**NOMBRE_DE_SU_FLUJO.**xml" http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1*

Si necesita eliminar el flujo cambie PUT por DELETE.

Compruebe la instalación del flujo.

Hito 10. Pruebe la conectividad con el flujo instalado. ¿Hay conectividad?. Explique por qué en cualquier caso. Capture con Wireshark el tráfico generado por un ping de h1 a h2. Explique la traza.

5.3. Parte II: Instalación de una aplicación en el controlador y prueba de funcionamiento. (Tiempo estimado de 4 horas)

1.7. Introducción

Antes de comenzar esta parte de la práctica, es conveniente saber que Opendaylight utiliza las siguientes herramientas de software:

- Interfaces Java: se utilizan como escuchadores de eventos, especificaciones y patrones formados. Gracias a ellas los *bundles* pueden realizar rellamadas en función de eventos
- Maven: se utiliza para compilar proyectos de forma automática. Utiliza ficheros pom.xml para moverse entre dependencias y saber que *bundles* ejecutar.
- OSGi: es el *framework backend* de Opendaylight. Permite cargar *bundles* y paquetes Jar, e intercambiar información entre ellos.

1.8. Maven

En la máquina virtual del controlador diríjase a la carpeta `/home/USER_X/intergration/tapapp` y ejecute el comando `mvn install -DskipTests -DskipIT -nsu`.

Cuando finalice la ejecución revise los errores.

Hito 11. Sustituya en todos los ficheros *pom.xml* el campo: group-id por la ruta necesaria para acceder a sus ficheros, con los niveles de carpetas separados por puntos. Vuelva a compilar el proyecto. No se olvide de incluir el nombre y la ruta de su *feature* en el fichero *features.xml*

Una vez compilada la aplicación, ejecute el controlador y compruebe si se ha instalado su *feature.* Para comprobarlo ejecute en el *prompt* de Karaf los siguientes comandos:

```
Opendaylight-user@root>feature:list | grep sdn
Opendaylight-user@root>bundle:list | grep sdn
```

1.9. Prueba de la aplicación

Para esta práctica la topología que vamos a emplear es la siguiente:

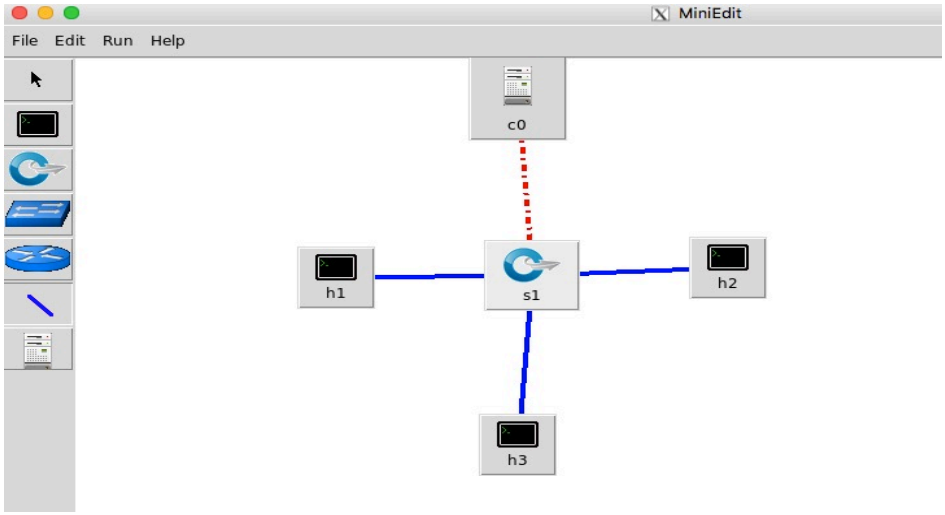


Figura 55: Topología manual parte 2

Fuente: Elaboración propia

Hito 12. En el terminal del controlador ejecute `log:tail | grep sdn`. Despliegue la topología conectada al controlador remoto y pruebe su conectividad. ¿Qué significan los mensajes que aparecen en el *log* del controlador?.

A continuación vamos a desarrollar un flujo JSON basado en nuestra aplicación.

Hito 13. Abra el fichero *TapProvider.java* que se encuentra en el directorio `/tapapp/implementation/src/main/java/`**`group-id`** y diga qué hace el *switch java* del método *private void addTap (Tap tap)*.

Hito 14. Teniendo en cuenta el hito anterior, rellene la plantilla que se muestra a continuación para realizar un flujo que filtre los paquetes ARP con origen el puerto 1 del *switch* y los reenvíe al puerto 2. Puede apoyarse en la interfaz web del controlador para conocer los id de los elementos de red.

```
{"tap-spec":

    {"tap":[
            {"id":"XXX",
                    "node":"openflow:XXX",
                    "traffic-match":"XXX",
                    "src-node-connector":["openflow:XXX:XXX"],
                    "sink-node-connector":["openflow:XXX:XXX"]
            }
            ]
    }
}
```

Hito 15. Desde el controlador envíe el flujo desarrollado al switch utilizando el comando *curl* explicado anteriormente. En el terminal con Mininet ejecute un *ping* del *host* 3 al *host* 1 y capture el tráfico con Wireshark en los tres *host*. Diga qué significan las trazas capturadas.

Elimine el flujo instalado en el switch y vaya a la interfaz web del controlador. En el menú de la izquierda desplácese a Yang UI. Dentro del menú anterior busque la entrada que hace referencia a la aplicación *tapapp* y despliegue la pestaña configuración.

Hito 16. Rellene los campos y seleccione *Send.* Vaya a Mininet y compruebe si el flujo se ha instalado. Realice capturas de tráfico de nuevo y compruebe el funcionamiento.

Hito 17 (opcional). En la topología desplegada por Mininet ejecute un servidor HTTP en el *host* 1. Configure el flujo de la aplicación para que filtre paquetes HTTP y haga una petición GET desde el *host* 3. Capture el tráfico con Wireshark y diga qué ocurre.

6. PLANIFICACIÓN Y PRESUPUESTO

En este punto se desglosan las tareas llevadas a cabo en este Trabajo Fin de Grado. Con este desglose se pretende facilitar los cálculos, tanto de horas como de costes.

6.1. Planificación

Las tareas que conforman este TFG se han divido en las siguientes fases ordenadas de forma cronológica:

Fase 1: Documentación inicial (50 horas)
1. Estudio de la arquitectura SDN (15 horas)
2.Estudio del protocolo OpenFlow (15 horas)
3.Estudio de Mininet (10 horas)
4.Estudio del controlador Opendaylight (10 horas)

Fase 2: Instalación y configuración de las máquinas virtuales (30 horas)
1.Instalación de la máquina virtual de Mininet (5 horas)
2.Instalación de la máquina virtual con sistema operativo Lubuntu 14 (15 horas)
3.Instalación del controlador Opendaylight (10 horas)

Fase 3: Pruebas de Mininet (20 horas)
1.Pruebas del entorno de trabajo de Mininet (10 horas)
2. Desarrollo de topologías personalizadas (10 horas)

Fase 4: Pruebas de Opendaylight (23 horas)
1.Pruebas con el controlador Opendaylight (15 horas)
2.Instalación de *features* para el controlador Opendaylight (8 horas)

Fase 5: Interconexión de ambas máquinas virtuales (30 horas)
1.Interconexión y prueba de conectividad entre máquinas virtuales (30 horas)

Fase 6: Desarrollo de la primera parte de la práctica (26 horas)
1.Desarrollo del flujo en XML (10 horas)
2.Instalación del flujo en el switch (6 horas)
3.Pruebas del funcionamiento del flujo instalado y corrección de errores (10 horas)

Fase 7: Desarrollo de la segunda parte de la práctica (53 horas)
1.Desacarga e instalación de la aplicación a utilizar (8 horas)

2.Creación y configuración del proyecto Maven para hacerlo funcionar en nuestro entorno (25 horas)

3.Pruebas de instalación de flujos basados en la aplicación (20 horas)

Fase 8: Elaboración de la memoria (68 horas)

1.Redacción de la memoria (60 horas)

2. Corrección y maquetación (8 horas)

En la tabla 1 se muestra un resumen de las tareas.

FASES	HORAS EMPLEADAS
Documentación inicial	50
Instalación de máquinas virtuales	30
Pruebas de Mininet	20
Pruebas de Opendaylight	23
Interconexión de máquinas virtuales	30
Desarrollo de la primera parte de la práctica	26
Desarrollo de la segunda parte de la práctica	53
Elaboración de la memoria	68
TOTAL	**300**

Tabla 1: Resumen de tareas del TFG

Fuente: Elaboración propia

A continuación se muestra un diagrama de Gantt de las fases del TFG y en el anexo A.2 encontramos otro con el total de tareas.

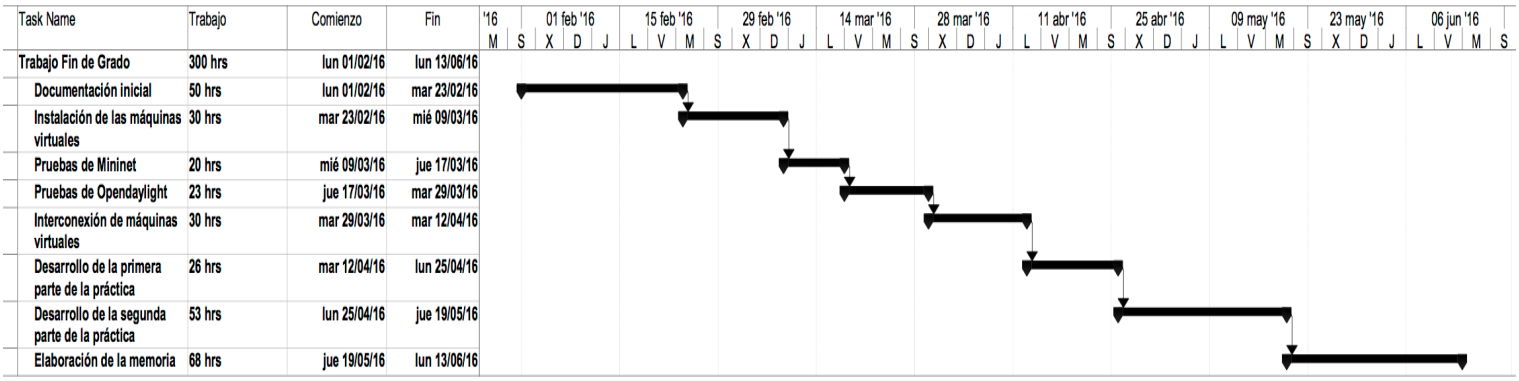


Figura 56: Diagrama de Gantt de fases del TFG

Fuente: Elaboración propia

6.2. Presupuesto

- **Autor:** José Manuel Frías Pérez
- **Departamento de Ingeniería Telemática**
- **Descripción del Proyecto**
  - → **Título:** Diseño y despliegue de una red definida por software
  - → **Duración:** 4'5 meses
  - → **Tasa de costes indirectos:** 20%
- **Presupuesto Total del Proyecto:** 13.188'662 €
- **Subcontratación de tareas:** No se especifican
- **Otros costes indirectos:** No se especifican

Concepto	Cantidad	Coste (€)	% Proyecto	Dedicación (meses)	Depreciación	Total (€)
Ordenador portátil	1	1000	100	4'5	60	75
TOTAL						**75**

Tabla 2: Costes materiales

Fuente: Elaboración propia

Siendo el coste imputable el resultado de la siguiente fórmula:

$$\text{CosteImputable} = \frac{A}{B} \text{ x } C \text{ x } D$$

Siendo los siguientes valores:

- A: Dedicación del equipo al TFG medido en meses:4 meses
- B: Período de depreciación del equipo: 60 meses.
- C: Coste del equipo: 1000 €
- D: Porcentaje de uso del equipo: 100% de dedicación.

$$\text{CosteImputable} = \frac{4{,}5}{60} \text{ x } 1000 \text{ x } 1 = 75 \text{ €}$$

Según un estudio [20] realizado por el Colegio Oficial de Ingenieros y la Asociación Española de Ingenieros de Telecomunicaciones, el salario medio de un ingeniero de telecomunicaciones es de 49.206 € anuales, mientras que para un ingeniero con menos de 5 años de experiencia es de 27.730 €. Además del coste deducido del estudiante en este Trabajo Fin de Grado se ha añadido el coste del tutor.

Concepto	Cantidad	Dedicación (ing/mes)	Salario Anual (€)	Total (€)
Ingeniero Senior	1	0,5	49.206	9.226’125
Ingeniero	1	0,37	27.730	3.847’537
TOTAL				13.113’662

Tabla 3: Coste de recursos humanos

Fuente: Elaboración propia

Concepto	Total (€)
Recursos Materiales	75
Recursos Humanos	13.113’662
TOTAL	13.188’662

Tabla 4: Coste total

Fuente: Elaboración propia

7. CONCLUSIONES Y TRABAJOS FUTUROS

En este punto se exponen las conclusiones al haber realizado este Trabajo Fin de Grado, así como posibles trabajos futuros que se podrían realizar para continuar este proyecto.

7.1. Conclusiones

Este Trabajo Fin de Grado se centra en el diseño y despliegue de una red definida por software sobre máquinas virtuales, con el objetivo de crear un guion de prácticas para alumnos de la Universidad Carlos III, y probar la utilidad de Mininet en el ámbito docente.

Para llegar a la solución final primero se han estudiado los controladores que existían y cual de ellos se adecuaba más a las necesidades del proyecto, inicialmente se utilizó el controlador Floodlight, pero su exigua documentación y la herramienta de trabajo proporcionada no era lo esperado.

Decidido el uso del controlador Opendaylight, se probó el software Mininet, la interconexión de ambas máquinas y se decidió la topología a utilizar. Desplegado el escenario SDN, se procedió a comprobar algunas de sus funcionalidades, instalando flujos y aplicaciones.

Hecho esto, se puede concluir que se han alcanzado todos los objetivos propuestos en este Trabajo Fin de Grado: Mininet se podría utilizar en los laboratorios de la universidad, es una herramienta de virtualización muy potente y versátil que permite desplegar topologías de red muy grandes con pocos recursos hardware, además, el CLI de la máquina virtual es familiar para cualquier estudiante introducido en el sistema operativo Linux.

Podemos añadir, que el estudio y la realización de prácticas sobre SDN en la universidad va a formar alumnos preparados para el futuro de las telecomunicaciones. Actualmente, muchas empresas están apostando por ello, y, hecho este Trabajo Fin de Grado, es fácil entender su potencial.

7.2. Trabajos futuros

A continuación se enumeran algunas de las posibles mejoras a realizar sobre este Trabajo Fin de Grado:

- **Ampliación de la topología desplegada en Mininet**: este TFG se ha realizado con una topología de red pequeña, en el futuro se podrían realizar pruebas con topologías mayores para ver el máximo potencial de Mininet.
- **Instalación de varios flujos en un switch**: se puede realizar la instalación de varios flujos en un mismo switch, para realizar aplicaciones reales, por ejemplo un *firewall*.
- **Mejora de la aplicación**: actualmente la aplicación instalada en nuestro controlador sólo permite visualizar los paquetes que utilizan el flujo instalado desde el propio switch. En soluciones futuras se podría mejorar la aplicación para que, utilizando el protocolo OpenFlow, el switch enviase al controlador el número de mensajes que le han llegado del protocolo que se esté filtrando.
- **Instalación de flujos en varios switches**: se podría realizar un escenario real utilizando varios switches e instalando flujos en todos ellos con distintas funcionalidades.
- **Utilización de la nueva versión del controlador Opendaylight**: debido a que el desarrollo de este TFG ha sido simultáneo al desarrollo y mejora del controlador Opendaylight, no se ha utilizado la última versión del mismo. En un futuro se podría utilizar la nueva versión y estudiar sus mejoras y cambios.

8. CONCLUSIONS AND FUTURE WORKS

In this section we will describe the findings from our project and suggest future works to continue the research.

8.1. Conclusions

This Final Bachelor Thesis focuses on the design and deployment of a software-defined network using virtual machines. The aim is to create a lab test guide for students from Carlos III University and test the usefulness of Mininet in the education environment.

For reaching the final solution we have first studied the existing controllers in order to find which one suited the project's needs better. Initially we used the Floodlight controller but the lack of information about it and the tool itself did not match our expectations.

Once we decided to use Openlight, we tested Mininet software, the interconnection of both machines, and which topology to use. When the SDN scene had been deployed, we tested some of its functionalities, installing streams and applications.

After all this, we can conclude that we achieved all the objectives we initially set for this Final Thesis: Mininet could be used in university labs. It is a powerful and versatile virtualization tool that allows to deploy wide network topologies with little hardware resources. In addition, the CLI from the virtual machine is familiar to any student introduced to Linux operating system.

The study and performance of tests about SDN at university is going to contribute to the training of students ready to approach the future of telecommunications. Many companies are betting for SDN and once we have finished this thesis, we can easily understand its potential.

8.2. Future works

Some of the improvements that could be made to this Final Bachelor Thesis are the following:

- **Widening of the topology deployed in Mininet**. This project was made with a relatively small network topology. In the future, tests could be made with bigger topologies in order to see Mininet's maximum potential.
- **Installation of several flows in a switch**: severals flows could be installed in a single switch to make real applications such as a firewall.
- **Improvement of the application.** Currently, the application installed in our Controller only allows the visualization of packages using the stream installed from the switch itself. In future solutions, the application could be improved so that the switch would send the number of messages received from the monitored protocol to the controller, using the OpenFlow protocol.
- **Installation of flows in various switches.** A real environment could be created by using several switches and by installing different flows with various functionalities.
- **Usage of the new version of the Opendaylight controller**. Due to the parallel development of Opendaylight and this Final Thesis, we have not used the last version of the controller. In the future, the new version could be used to study its changes and improvements.

ANEXOS

En este punto se añadirá información adicional sobre ciertos puntos de este Trabajo Fin de Grado.

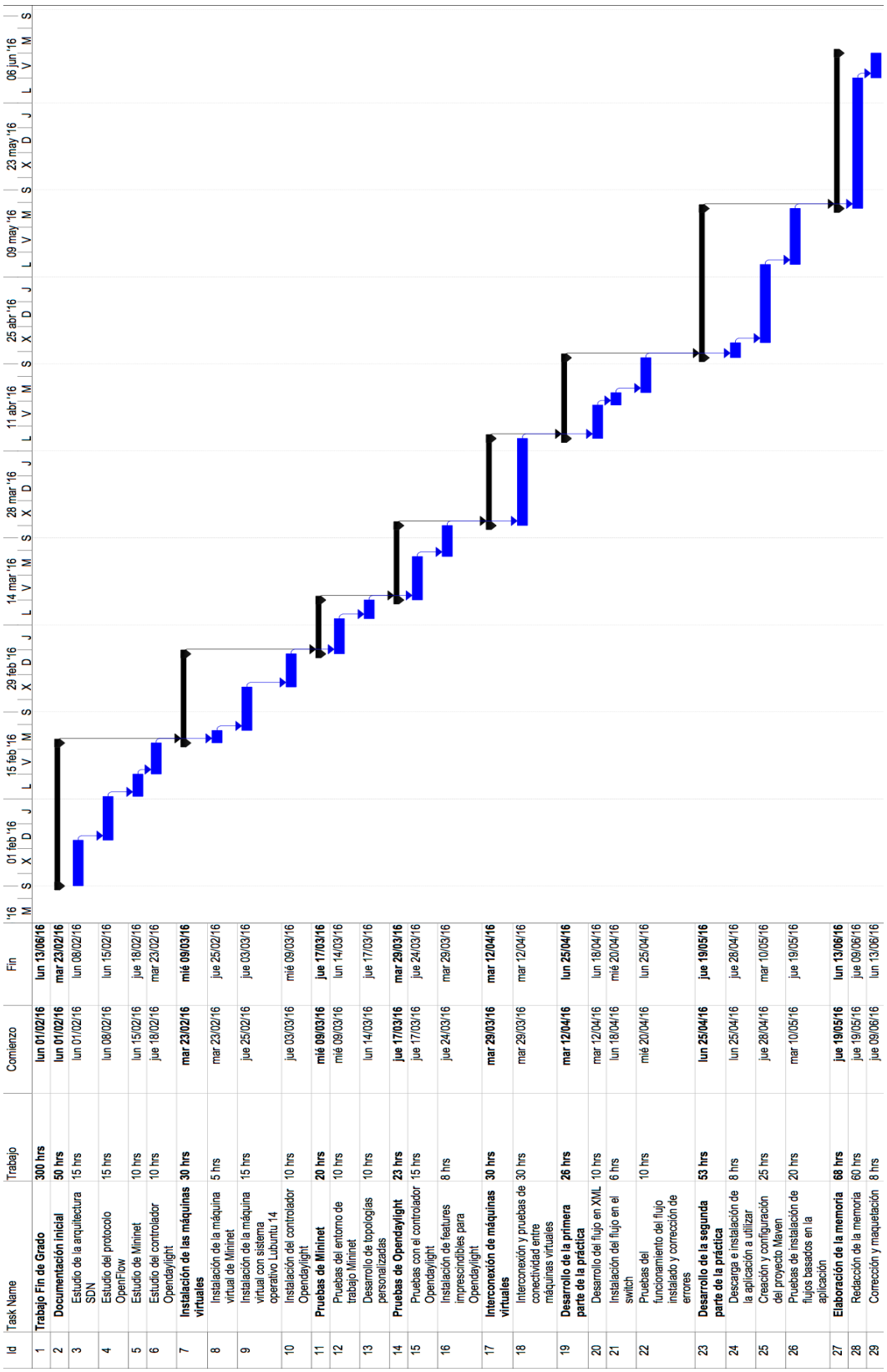
A.1 Marco Regulador

Actualmente no existe ningún marco regulador que pueda ser aplicado al desarrollo de este Trabajo Fin de Grado.

Software empleado:

- Virtual Box: licenciado como *General Public License*.
- Ubuntu 14.04 LTS licenciado como *General Public License*.
- OS X El Capitan 10.11 licenciado como *Apple Public Source License*.
- Mininet licenciado como *BSD Open Source license*.
- Opendaylight licenciado como *Eclipse Public License.*

A.2 Diagrama de Gantt



Id	Task Name	Trabajo	Comienzo	Fin
1	**Trabajo Fin de Grado**	**300 hrs**	**lun 01/02/16**	**lun 13/06/16**
2	**Documentación inicial**	**50 hrs**	**lun 01/02/16**	**mar 23/02/16**
3	Estudio de la arquitectura SDN	15 hrs	lun 01/02/16	lun 08/02/16
4	Estudio del protocolo OpenFlow	15 hrs	lun 08/02/16	lun 15/02/16
5	Estudio de Mininet	10 hrs	lun 15/02/16	jue 18/02/16
6	Estudio del controlador Opendaylight	10 hrs	jue 18/02/16	mar 23/02/16
7	**Instalación de las máquinas virtuales**	**30 hrs**	**mar 23/02/16**	**mié 09/03/16**
8	Instalación de la máquina virtual de Mininet	5 hrs	mar 23/02/16	jue 25/02/16
9	Instalación de la máquina virtual con sistema operativo Lubuntu 14	15 hrs	jue 25/02/16	jue 03/03/16
10	Instalación del controlador Opendaylight	10 hrs	jue 03/03/16	mié 09/03/16
11	**Pruebas de Mininet**	**20 hrs**	**mié 09/03/16**	**jue 17/03/16**
12	Pruebas del entorno de trabajo Mininet	10 hrs	mié 09/03/16	lun 14/03/16
13	Desarrollo de topologías personalizadas	10 hrs	lun 14/03/16	jue 17/03/16
14	**Pruebas de Opendaylight**	**23 hrs**	**jue 17/03/16**	**mar 29/03/16**
15	Pruebas con el controlador Opendaylight	15 hrs	jue 17/03/16	jue 24/03/16
16	Instalación de features imprescindibles para Opendaylight	8 hrs	jue 24/03/16	mar 29/03/16
17	**Interconexión de máquinas virtuales**	**30 hrs**	**mar 29/03/16**	**mar 12/04/16**
18	Interconexión y pruebas de conectividad entre máquinas virtuales	30 hrs	mar 29/03/16	mar 12/04/16
19	**Desarrollo de la primera parte de la práctica**	**26 hrs**	**mar 12/04/16**	**lun 25/04/16**
20	Desarrollo del flujo en XML	10 hrs	mar 12/04/16	lun 18/04/16
21	Instalación del flujo en el switch	6 hrs	lun 18/04/16	mié 20/04/16
22	Pruebas del funcionamiento del flujo instalado y corrección de errores	10 hrs	mié 20/04/16	lun 25/04/16
23	**Desarrollo de la segunda parte de la práctica**	**53 hrs**	**lun 25/04/16**	**jue 19/05/16**
24	Descarga e instalación de la aplicación a utilizar	8 hrs	lun 25/04/16	jue 28/04/16
25	Creación y configuración del proyecto Maven	25 hrs	jue 28/04/16	mar 10/05/16
26	Pruebas de instalación de flujos basados en la aplicación	20 hrs	mar 10/05/16	jue 19/05/16
27	**Elaboración de la memoria**	**68 hrs**	**jue 19/05/16**	**lun 13/06/16**
28	Redacción de la memoria	60 hrs	jue 19/05/16	jue 09/06/16
29	Corrección y maquetación	8 hrs	jue 09/06/16	lun 13/06/16

Figura 57: Gantt completo

Fuente: Elaboración propia

Anexo A.3 Flujo XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
        <instructions>
                <instruction>
                        <order>0</order>
                        <apply-actions>
                                <action>
                                  <order>0</order>
                                <drop-action/>
                                </action>
                        </apply-actions>
                </instruction>
        </instructions>
                <table_id>0</table_id>
                <id>1</id>
                <cookie_mask>255</cookie_mask>
                <installHw>false</installHw>
                <match>
                  <ethernet-match>
                  <ethernet-destination>>
                        <address>00:00:00:00:00:02</address>
                  </ethernet-destination>
                  <ethernet-source>
                        <address>00:00:00:00:00:01</address>
                  </ethernet-source>
                  </ethernet-match>
                </match>
                <hard-timeout>120</hard-timeout>
                <cookie>3</cookie>
                <idle-timeout>34</idle-timeout>
                <flow-name>Parte1</flow-name>
                <priority>15</priority>
                <barrier>false</barrier>
</flow>
```

Anexo A.4 Flujo JSON

{"tap-spec":

{"tap":[
{"id":"1",
"node":"openflow:1",
"traffic-match":"ARP",
"src-node-connector":["openflow:1:1"],
"sink-node-connector":["openflow:1:2"]
}
]
}
}

Anexo A.5 Solución del guion de prácticas

Hito 1.

- `sudo mn:` Despliega la topología por defecto de Mininet.
- `nodes:` Muestra los nodos de la red desplegada.
- `net:` Muestra los enlaces de la red.
- `dump:` Muestra información de todos los nodos.
- `h1 ifconfig -a:` Muestra las interfaces de red del host 1.
- `s1 ifconfig -a:` Muestra las interfaces de red del switch 1.
- `pingall:` Realiza un ping entre todos los host de la red.
- `xterm h1:` Ejecuta una ventana adicional para trabajar directamente con el host 1.

Las diferencias que encontramos entre los dos comandos *ifconfig* realizados son: al ejecutar el comando en el *host* 1, sólo se muestran interfaces virtuales generadas por Mininet, mientras que si lo hacemos en s1, la respuesta entrega tanto interfaces virtuales como reales del *host* anfitrión. Esto es debido a que s1 se ejecuta en el espacio de nombres de red, *root*. El resultado es el mismo qué si ejecutamos *ifconfig* en el sistema operativo Linux anfitrión.

Hito 2. `sudo mn --controller=remote,ip=[controller ip],port=[controller listening port]`

Hito 3. En la interfaz web del controlador aparecerá el *switch* de la topología desplegada. Al realizar el comando *pingall* el controlador aprende la topología total desplegada. Ve tanto los *switches* como los *hosts*. Esto se debe a la instalación de flujos por parte del controlador.

Hito 4. Al ejecutar el comando por primera vez no se ve ningún flujo instalado. Al hacer *pingall* el control de tráfico del controlador instala los flujos necesarios para conocer la topología de red.

Hito 5. La segunda vez que se ejecuta el comando *ping* no se aplica el control de tráfico del controlador debido a que ya se han aplicado los flujos ICMP, por ello el tiempo de llegada del segundo *ping* es más corto. Los paquetes pasan directamente por el *switch* sin pasar por el controlador.

Hito 6, 7, 8.

```
    net = Mininet( topo=None,listenPort=6633,
            build=False,
            ipBase='10.0.0.0/8')

info( '*** Adding controller\n' )
c0=net.addController(name='c0',
                controller=RemoteController,
                protocol='tcp', protocols='OpenFlow13',
                ip='1.1.1.2'
                )

info( '*** Add switches\n')
s3 = net.addSwitch('s3', cls=OVSKernelSwitch,          mac='00:00:00:00:00:06',
        protocols='OpenFlow13')
s2 = net.addSwitch('s2', cls=OVSKernelSwitch,          mac='00:00:00:00:00:05',
        protocols='OpenFlow13')
s1 = net.addSwitch('s1', cls=OVSKernelSwitch,          mac='00:00:00:00:00:04',
        protocols='OpenFlow13')

info( '*** Add hosts\n')
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3',  defaultRoute=None,
        mac='00:00:00:00:00:03')
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2',  defaultRoute=None,
        mac='00:00:00:00:00:02')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1',  defaultRoute=None,
        mac='00:00:00:00:00:01')

info( '*** Add links\n')
net.addLink(s2, s1)
net.addLink(s2, s3)
net.addLink(s3, h3)
net.addLink(s2, h2)
net.addLink(s1, h1)
```

```
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s3').start([c0])
    net.get('s1').start([c0])
    net.get('s2').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Para comprobar la conectividad ejecutamos el comando *pingall.*

Hito 9. Solución en el anexo A.3

Hito 10. No hay conectividad entre h1 y h2, debido a que al instalar el flujo anterior los paquetes con origen la MAC de h1 y destino la MAC de h2 son descartados.

Hito 11. Ejemplo de fichero *pom* corregido:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <parent>
      <groupId>uc3m.sdn.odl.tfg</groupId>
      <artifactId>commons</artifactId>
      <version>1.2.0-SNAPSHOT</version>
      <relativePath>commons/parent</relativePath>
    </parent>

    <artifactId>main</artifactId>
    <version>0.6.0-SNAPSHOT</version>
```

```
    <packaging>pom</packaging>
    <url>http://sdnhub.org/tutorials/opendaylight</url>
    <modules>
      <module>commons/parent</module>  <!-- Common properties -->
      <module>commons/utils</module>   <!-- Common utilities useful for flow
programming -->
      <module>tapapp</module>          <!-- App 2: Traffic tap -->
      <module>features</module>        <!-- Karaf feature description -->
      <module>distribution</module>    <!-- Branding and packaging -->
    </modules>

</project>
```

Aplicación añadida al fichero *features.xml*

```
<feature name="sdn-tapapp" description="SDN :: OpenDaylight :: Tap application"
version='${project.version}'>
    <feature version="${openflowplugin.version}">odl-openflowplugin-
southbound</feature>
    <feature version="${openflowplugin.version}">odl-openflowplugin-flow-
services</feature>
    <feature version="${feature.mdsal.version}">odl-mdsal-broker</feature>
    <bundle>mvn:uc3m.sdn.odl.tfg.tapapp/tapapp-impl/1.0.0-SNAPSHOT</bundle>
    <bundle>mvn:uc3m.sdn.odl.tfg.tapapp/tapapp-model/1.0.0-SNAPSHOT</bundle>
    <configfile
finalname="etc/opendaylight/karaf/${tapapp.configfile}">mvn:uc3m.sdn.odl.tfg.tapapp
/tapapp-config/1.0.0-SNAPSHOT/xml/config</configfile>
    </feature>
    </features>
```

Hito 12. Los mensajes del *log* muestran la nueva conexión de una topología de red al controlador, y nos proporciona información sobre los elementos de la misma.

Hito 13. El *switch java* se encarga de traducir los códigos de los protocolos a lenguaje natural, para que sea más sencillo filtrar cada uno de ellos.

Hito 14. Solución en el anexo A.4

Hito 15. Al instalar el flujo todos los paquetes con origen el puerto 1 del *switch* son enviados al puerto 2. De este modo la conectividad entre h3 y h1 se pierde. H3 emite un

mensaje *ARP Request* pero la respuesta *ARP Reply* de h1 no llega a h3. Tanto h1 como h3 tras varios intentos, comienzan a emitir a *Broadcast*.

Hito 16. El flujo se instala correctamente. Las capturas son iguales a las del hito 15.

Hito 17. La petición GET llega a h1 pero la respuesta emitida por este no llega nunca a h3. Es h2 el que recibe dicha respuesta.

BIBLIOGRAFÍA

[1] Open Networking Foundation (Abril 2012). Software-Defined Networking: The New Norm for Networks. Última consulta: 24/05/2016. Recuperado de: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[2] CISCO (2014) Software Defined Networking: en busca de la automatización de la red. Última consulta: 20/05/2016. Recuperado de: https://www.cisco.com/web/mobile/global/la/ofertas/fastit/pdfs/fs_white_paper_cisco.pdf

[3] Mininet Team (2016) Mininet [Software] Disponible en: http://mininet.org/

[4] CITRIX (Mayo 2014). SDN 101: Introducción a Software Defined Networking. Última consulta: 20/05/2016. Recuperado de: https://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking-es.pdf

[5] SDxCentral (2015) What's Software Defined Networkig (SDN)? Definition. Última consulta: 22/05/2016. Recuperado de: https://www.sdxcentral.com/sdn/resources/what-the-definition-of-software-defined-networking-sdn/

[6] Nunes B., Mendonca M., NguyenX., Obraczka K., Turletti,T. et al. (2014) *A survey of software-defined networking: Past, present, and future of programmable networks*. Communications Surveys & Tutorials, IEEE, 16(3): 1617–1634 Última consulta: 19/05/2016. Recuperado de: https://hal.inria.fr/hal-00825087v5/document

[7] McKeown K., Anderson T., Balakrishnan, H., Parulkar G., Peterson L., Jennifer Rexford, Shenker S. y Turner J. (Marzo, 2008) *OpenFlow: Enabling Innovation in Campus Networks*. Última consulta: 27/05/2016. Recuperado de: http://archive.openflow.org/documents/openflow-wp-latest.pdf

[8] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review 2008. Mayo-2013. Última consulta: 11/06/2016. Recuperado de: http://slideplayer.com/slide/7286741/

[9] Open Networking Foundation (Junio, 2012) *Openflow switch specification 1.3*. Última consulta: 01/06/2016. Recuperado de: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[10] Linux Foundation (2016) Opendaylight Controller [Software] Disponible en: https://www.opendaylight.org/

[11] Sdxcentral (2012-2016) What is an Opendaylight Controller? AKA: Opendaylight Platform. Última consulta: 24/05/2016. Recuperado de: https://www.sdxcentral.com/sdn/resources/sdn-controllers/opendaylight-controller/

[12] Sdn Tutorials (2013) Difference between ad sal and md sal. Última consulta: 11/05/2016. Recuperado de: http://sdntutorials.com/difference-between-ad-sal-and-md-sal/

[13] Velásquez Vargas, W.A. (2014). *Emulación de una red definida por software utilizando mininet*. Última consulta: 26/05/2016. Recuperado de: http://www.academia.edu/5730624/Emulaci%C3%B3n_de_una_red_definida_por_software_utilizando_MiniNet

[14] Ryu SDN Framework Community (2014) RYU (Información sobre Controlador) Última consulta: 27/05/2016. Recuperado de: https://osrg.github.io/ryu/

[15] Project Floodlight.Org (2016) Project Floodlight (Información sobre Controlador) Última consulta: 27/05/2016. Recuperado de: http://www.projectfloodlight.org/floodlight/

[16] Nicira Networks (2012) Noxrepo (Información sobre Controlador) Última consulta: 27/05/2016. Recuperado de: http://www.noxrepo.org/

[17] Open vSwitch (2014) What is Open vSwitch? Última consulta: 03/06/2016. Recuperado de: http://openvswitch.org/

[18] Opendaylight. Org (2014) Install on Ubuntu 14.04 Última consulta: 03/04/2016. Recuperado de: https://wiki.opendaylight.org/view/Install_On_Ubuntu_14.04

[19] SDN Hub (Mar 2016) SDN Hub Opendaylight Tapapp Applications. Última consulta: 12/05/2016. Recuperado de: https://github.com/sdnhub/SDNHub_Opendaylight_Tutorial/tree/master/tapapp

[20] ABC TECNOLOGÍA (Feb 2013) Nueve de cada diez ingenieros de telecomunicaciones tienen empleo. Última consulta: 13/06/2016. Recuperado de: http://www.abc.es/tecnologia/noticias/20130219/abci-trabajo-ofertas-telecomunicaciones-201302181108.html